A PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SOLO PROGRAMADORES

O IV, NÚMERO 43

NÚMERO ESPECIAL 1.450 Ptas.

NTELIGENCIA ARTIFICIAL

COMPUTACIÓN EVOLUTIVA



PROGRAMACIÓN DE JUEGOS
El uso del tiempo

SISTEMAS DISTRIBUIDOS
Run Time System

VISUAL C++
Mapas de mensajes

SISTEMAS OPERATIVOS
Funciones hardware de micros Intel

REDES LOCALES
Cactus 3.0

SERVIDORES
Unix con Samba

PROFESIONALES
Simulación de sistemas

TOWER

8 413042 303299

CONTENIDO DEL CD-ROM

- DB2 Universal Database 5.0
- Hyperwire v1.0 Test Drive
- McAfee NetXRay 3.0
- Utilidades de entornos distribuidos de McAfee
- ICQ 98a
- Bases de datos para Unix
- Utilidades para el cambio de milenio
- WordPerfect Linux 7.0
- Y mucho más...

SÓLO PROGRAMADORES

Número 43
SÓLO PROGRAMADORES
es una publicación de
TOWER COMMUNICATIONS

Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Coordinadora de Redacción
Silvia Galán Sicilia
sgalan@towercom.es
Colaboradores
Enrique Díaz, Eugenio Castillo Jiménez, Ernesto Schmitz, Chema Alvarez, J.Carlos Gómez, Constantino Sánchez, Javier Toledo Fernández, Jorge Figeroa, Xesco Hernández, Juan J. Taboada, José J. Mora, Manuel de la Herrán Gascón, Jorge F. Delgado, Antonio J. Novillo
Maquetación
Susana Llano
Tratamiento de Imagen
María Arce Giménez

Publicidad
Erika de la Riva (Madrid)
Tel.: (91) 661 42 11
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Isabel Bojo
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Laboratorio
Óscar Rodríguez (jefe)
oscarrf@towercom.es
Javier Amado
jamado@towercom.es
Servicio Técnico
Óscar Casado
ocasado@towercom.es

Filmación
Megatipo
Impresión
Gráficas Reunidas
Distribución
SGEL
Distribución en Argentina
Capital: Huesca y Sanabria
Interior: D.G.P.

TOWER COMMUNICATIONS
Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
carmina@towercom.es

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 31-05-98

EDITORIAL

La máquina que llora

En su larga cadena de superación, el hombre tiene todavía, a estas alturas, un eslabón pendiente: fabricar una máquina que sea capaz de llorar. Mientras los científicos se afanan en su irrevocable determinación de llevar a la práctica la clonación de los seres humanos, los informáticos estamos "enredados" en el intento de conseguir una máquina con características humanas... O superiores. En cuanto a las posibilidades creativas, ahí están los experimentos de programas como Tierra o la fulminante victoria de Deep Blue sobre el campeón Kaspárov. Su futuro y su significación están aún por determinar, pero no hay duda de que son importantes pasos en ese camino. En cuanto a hacer "comprender" a una máquina nociones humanas, como las sensaciones del placer y el dolor, el asunto es ya algo más complejo. Dejando a los filósofos la tarea de dirimir las implicaciones de todo ello, para nosotros es sencillamente un problema difícil de resolver. Además, si una máquina llega a conocer las lágrimas, probablemente lo primero que hará será llorar su nueva condición.

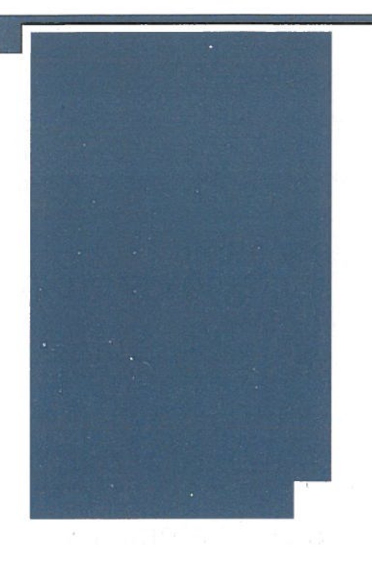
La evolución de la Inteligencia Artificial y su problemática son analizadas en las páginas de nuestro artículo central. El autor del mismo, haciéndose eco de la evidente preocupación social acerca de este fascinante tema (recordemos tan sólo el impresionante impacto de filmes como *2001, una odisea en el espacio* y *Blade Runner*, o incluso la evolución de los androides en las conocidas sagas *Star Trek* y *Alien*), muestra con acierto los principales motores de dicha evolución y las distintas teorías que la explican, desde la pragmática lucha por la supervivencia que expuso Charles Darwin hasta el romántico apoyo mutuo que el príncipe ruso Pedro Kropotkin hizo famoso entre los revolucionarios de su época.

Se trata, sin duda, de un tema apasionante, pero muchos otros han sido analizados dentro de este número. El programa Cactus 3.0 se encuentra en el centro de atención del apartado que está dedicado a las redes locales; continuamos profundizando en el lenguaje de programación Visual C++ con algunas macros de mapas de mensajes; echamos un vistazo a la programación de DirectX por medio de Delphi en su versión 2.0; y, por último, en nuestra serie de artículos sobre los sistemas distribuidos vemos en esta ocasión el Run Time System.

Tenéis a vuestra disposición la dirección de correo electrónico de la revista: solop@towercom.es, donde podéis enviar vuestras sugerencias sobre los contenidos, los temas que queráis que incluyamos y los productos que os gustaría que distribuyéramos con el CD-ROM, que todos los meses regalamos. Sois nuestros mejores colaboradores.

SÓLO PROGRAMADORES





Noticias

Las estaciones de trabajo UNIX no desaparecen

ULTRA 5 Y ULTRA 10 DE SUN, ESTACIONES DE TRABAJO UNIX DE GRAN POTENCIA A PRECIOS DE PCS

La empresa Sun ha presentado recientemente una nueva gama de soluciones para informática técnica que incluye las nuevas estaciones de trabajo Ultra 5 y Ultra 10. Estas estaciones demuestran que eran exageradas las predicciones de que el auge de las estaciones de trabajo UNIX habían pasado a la historia.

Según Peter Foulkes (Dataquest), mediante la combinación de la superioridad tecnológica y escalabilidad de los sistemas basados en UNIX/RISC con la economía a gran escala del modelo Intel/Microsoft, Sun asume una vez más su posición de liderazgo en el campo de la relación precio/rendimiento para la informática técnica. Estos sistemas harán que muchos usuarios de UNIX pongan en entredicho la migración a Windows NT en un momento dado.

Jim Garden (Technology Business Research) sostiene que Sun ha presentado su oferta de estaciones de trabajo con una significativa propuesta de valor. Sun va a seguir en lucha con los fabricantes de estaciones UNIX tradicionales que compiten con los equipos Ultra 5 y Ultra 10 mediante la oferta de gráficos, servicios y soporte de estaciones de trabajo superiores a precios de PCs.

Las estaciones de trabajo Ultra tienen una alta potencia y un bajo precio. Persiguen el fin de captar la cuota del creciente mercado de PCs de la gama alta o estaciones de trabajo personales. La línea Darwin de Sun, en la que se incluyen ambas estaciones, combinan aspectos de las estaciones de trabajo Sun de alto rendimiento (gráficos atractivos y aplicaciones de productividad) con el nivel de precios y fabricación propios del sector de los PCs.

Con un precio inicial menor que el de muchos PCs, Sun se dirige de forma activa al creciente número de usuarios con grandes necesidades de rendimiento pero que, además, quieren los precios y compatibilidad de los PCs, con las aplicaciones de PCs más conocidas. Para ellos, Sun ofrece la potencia, las aplicaciones sofisticadas y los gráficos de gran precisión que ofrecen las estaciones de trabajo SPARC Solaris.

El auténtico caballo de batalla de Sun ha sido, pues, el de conseguir una bajada en los costes. Para reducir los precios, los competidores de Sun han tenido que pasar a PCs que ejecutan NT. Sin embargo, se ha optado por combinar la tecnología de diseño de sistemas VLSI con la fabricación a gran escala característica de los PCs, para conseguir la ansiada reducción.

La estación de trabajo Darwin de Sun se encuentra disponible en dos modelos diferentes. La estación Ultra 5, dirigida a usuarios que quieren un rendimiento de estación de trabajo a un precio asequible, y la estación de trabajo Ultra 10, diseñada para usuariosque necesitan más rendimiento, gráficos más potentes y mayores opciones de expansión.

MICROSOFT SQL SERVER 6.5 ENTERPRISE EDITION

Actualmente se encuentra disponible Microsoft SQL Server 6.5 Enterprise Edition, integrado con NT Server Enterprise Edition. Extiende las prestaciones y la escalabilidad disponible sobre el S.O. Windows NT. Especialmente interesante para las aplicaciones corporativas es el nivel de disponibilidad obtenido con la tecnología *cluster* a partir de elementos hardware estándar, lo que redunda en un coste muy competitivo. Aporta, además, una plataforma idónea para desarrollar de una forma sencilla aplicaciones críticas y distribuidas con acceso a datos. En este sentido, SQL Server 6.5 Enterprise Edition se beneficia de una perfecta integración con Microsoft Transaction Server (MTS) y Microsoft Message Queue (MSMQ), como con los demás componentes de BackOffice.

Microsoft SQL Server 6.5 Enterprise Edition incorpora soporte para servidores Symmetric Multi Processing (SMP) con hasta ocho procesadores y está previsto que, próximamente, esté disponible con 32 procesadores a través de soluciones OEM de fabricantes hardware *partners* de Microsoft. El soporte para cluster de alta disponibilidad Microsoft Cluster Server (MSCS), basado en la tecnología "WolfPack" de dos nodos de Microsoft, proporciona 1000% de protección frente a fallos del hardware, gestionando de forma automática la disponibilidad de la aplicación tras la caída o la recuperación de un nodo. También se incorpora en esta versión el soporte VLM para el direccionamiento de hasta 3 Gb. de memoria RAM, sólo disponible sobre procesadores Intel 32 bit. Próximamente estará dispoonible el soporte de Very Large Memory (VLM) sobre procesadores 64 bit Alpha Digital. Como última novedad Microsoft presenta también con este producto su interfaz de lengauaje natural, que permite acceder a la información a través de consultas formuladas en inglés en lugar del lenguaje formal de consultas tradicional, SQL.

PRIMER FORO DE AYUDA ON-LINE PARA DESARROLLADORES

Lotus Development Corporation ha puesto en marcha un centro de ayuda *on-line* especialmente dirigido a desarrolladores de software y de sedes web, vía Internet. Se trata de una experiencia pionera por la amplitud de la información y servicios a los que podrán acceder los desarrolladores.

La sede de Lotus es accesible a través de http://www.lotus.com y, además de ofrecer un foro de discusión para toda la comunidad de desarrolladores, ofrece trucos, herramientas, ejemplos de soluciones interactivas, descargas de software, documentación técnica muy detallada, artículos, referencias, presentaciones y demostraciones realizadas en cursos y seminarios específicos y suscripción a revistas especializadas. En las únicas dos áreas que se necesita un registro previo, por parte de todos los usuarios, es en las áreas de descarga y en el área de trucos de la semana.

Las tecnologías Domino y Notes, así como la línea de desarrollo de aplicaciones en Java, Lotus eSuite DevPack, junto con otras herramientas complementarias para desarrollar soluciones interactivas, son las protagonistas de esta sede, llamada Lotus Developer Central. Para todos aquéllos desarrolladores que estén interesados en certificarse como expertos en tecnología Lotus, la sede dispone de un apartado específico para ello.

Breves

I CONCURSO IMAGINACIÓN FACTORY

La empresa www.imaginacion.com, dedicada a la comercialización de la recopilación de todo lo necesario para la creación y el diseño de páginas web, ha convocado el I Concurso Imaginación Factory en el que se repartirán más de 150.000 ptas. en premios. El plazo para los participantes se cierra el día 30 de mayo Todas las condiciones y requisitos de la convocatoria están disponibles en (http://www.imaginacion.com). Todas las imágenes y archivos que presenta esta empresa están estructurados para facilitar al máximo el trabajo de cualquier diseñador de páginas web, ya sea profesional o no, pudiendo contar con más de 50.000 imágenes, 9.000 botones, 6.000 fondos y texturas, etc.

ESUITE WORKPLACE Y ESUITE DEVPACK DE LOTUS, APLICACIONES DE LOTUS BASADAS EN EL ESTÁNDAR JAVA

WorkPlace es la primera solución de productividad completamente basada en Java. Se trata de un escritorio compacto y fácil de usar desde donde se puede acceder rápidamente al correo electrónico y al navegador, así como a un conjunto de aplicaciones electrónicas de productividad comercial. Incluye tratamiento de textos, hoja de cálculo, correo electrónico, planificador de citas, etc.

DevPack proporciona a los desarrolladores de aplicaciones web las applets Java necesarios para crear applets interactivas de valor estratégico. La versión preliminar, que estará en veintiún idiomas, se puede descargar gratuitamente de la web. DevPack Lotus está preparando un amplio programa de formación a desarrolladores para estas aplicaciones.

SOPORTE A SUN DE INTEL EN LA MIGRACIÓN DE SOLARIS A LA ARQUITECTURA IA-64

Sun e Intel han anunciado un acuerdo para optimizar el entorno operativo Solaris de Sun para el futuro procesador Merced de Intel. Sun dispondrá de una versión optimizada de 64 bits de Solaris para Merced, en el momento en que se encuentren disponibles los primeros sistemas basados en Merced, previstos para 1999.

En virtud de este acuerdo, Intel y Sun colaborarán en la migración y optimización de todo el entorno operativo Solaris para sistemas basados en Merced. Además, Sun también ha anunciado un centro de migración de "Solaris para Intel", que tiene como fin ayudar a que los ISVs (fabricantes de software) y OEMs (fabricantes de hardware) ajusten y optimicen aplicaciones para Solaris sobre plataforma Intel. Como parte del programa, Intel presentará su soporte técnico. Asimismo, las dos compañías se esforzarán al máximo en las iniciativas de la plataforma de servidores IA-32 e IA-64.

La combinación de Solaris y Merced ofrecerá una opción potente a los clientes de IA-64. Además, facilitará una plataforma estratégica cuyo fin es la oferta de una conexión de red constante y global que conecte a los empleados, clientes y socios de una compañía.

Como primer miembro de la familia de microprocesadores de 64 bits de Intel, el procesador Merced ampliará la arquitectura Intel con nuevos niveles de rendimiento y funcionalidades para servidores y estaciones de trabajo, además de mantener plena compatibilidad con versiones anteriores de la familia de productos IA-32 de Intel.

Sun se compromete a ampliar el uso de Solaris en las plataformas informáticas líderes de Intel, así como en las plataformas SPARC. Confirmando esa dirección, se encuentra la reciente selección de Solaris por parte de NCR Corporation al elegir su sistema operativo UNIX.

Al estar diseñado para profesionales de la tecnología de la información, responsables de líneas de negocio, proveedores de servicios Internet (ISPs) y usuarios que precisen gran potencia informática, cualquier tipo de dispositivo conectado a la red puede acceder y utilizar Solaris, incluyendo NCs, NetPCs, ordenadores basados en Windows y servidores NT.

INFRESCO PRESENTA LA NUEVA VERSIÓN DE OPAL 2.1

La nueva versión de Opal 2.1, que ha sido presentada recientemente por Infresco (compañía subsidiaria del gigante del software Computer Associates , CA), es una nueva herramienta de integración que contiene múltiples finalidades combinando aplicaciones empresariales de diversas fuentes. Ofrece un interfaz multimedia de usuario mejorado y pudiéndose implementar en diversos entornos.

El mercado de herramientas que integran e implantan aplicaciones en cualquier lugar, según los analistas, alcanzarán el billón de dólares en el año 2000.

Opal proporciona potentes capacidades de integración entre fuentes de información de bases de datos y *mainframes.*

El producto amplía la funcionalidad de las actuales aplicaciones basadas en *host* y, a su vez, permite la migración invisible al entorno cliente/servidor. De forma simultánea, Opal crea un interfaz de usuario atractivo e intuitivo que reside en un nivel superior al del proceso.

Opal 2.1 integra un conjunto mejorado de herramientas interoperable. Entre ellas figuran: Opal Integrator, entorno de diseño e integración que opera en Windows 95 y NT; Opal Server, que ofrece comunicación con *hosts* y bases de datos en Windows NT; y Opal Player, la parte cliente de ejecución que opera con todas las versiones de Windows como ejecutables independientes, y en los navegadores Nescape Navigator e Internet Explorer.

A medida que, poco a poco,los analistas de la industria examinan vías para que las diferentes empresas implementen herramientas de integración, orientadas a la conexión de recursos heredados con la tecnología web actual y futura, otorgan una consideración especial a Opal.

Computer Associates distribuye Opal a través de su fuerza de venta doméstica e internacional, de manera directa e indirecta.

También se está apoyando los esfuerzos realizados por CA con acciones de marketing y formación, así como a través de sus propios canales directos de distribución.

Infresco en la actualidad está ofreciendo cursos de formación sobre el producto y visitando las diferentes instalaciones de los clientes a nivel internacional para proporcionar programas piloto de formación e implementación.

LIBROS

Curso de WebMaster

El curso WebMaster de Programación Internet, que ha sacado a la venta Tower Communications, forma verdaderos expertos en esta área ofreciendo los conocimientos necesarios para dominar la programación tanto para Intranets como para Internet.

Dicho curso ofrece una amplia y detallada información en lenguajes y programas como Active X, FPT, Java Script, FrontPage, CGI, HTML, Visual Basic, entre otros.

Internet es un importante mercado de trabajo en el que existe una amplia y creciente oferta de puestos dada la casi nula existencia de auténticos expertos en este sector. Por ello, Tower Communications cubre el importante vacío existente en un campo donde crece la demanda de personal realmente cualificado en la materia.

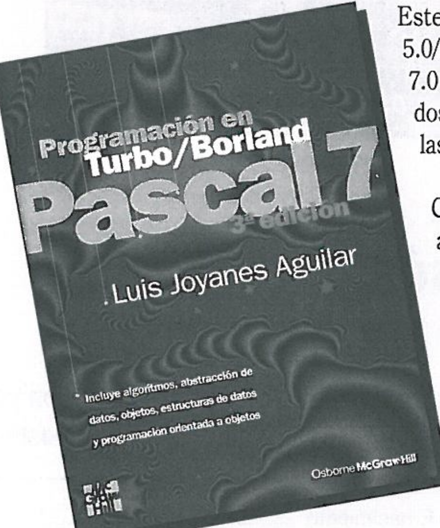
Al finalizar el curso todos los alumnos recibirán un diploma acreditativo de titulación y participarán en el Primer Concurso de Diseño de Currículums Interactivos. Los ganadores tendrán la oportunidad de trabajar creando páginas webs para las empresas patrocinadoras (Micronet, Sistemas Modernos de Marketing, Grupo Albertina de Comunicación, Abeto Editorial y Tower Communications).

Editorial: Tower Communications
Incluye: 50 fascículos y 5 CD-ROMs
Nivel: medio

Autor: Jorge Ferrer y F.E Valderrama
Idioma: castellano
Precio: 750 por entrega



Programación en Turbo/Borland Pascal 7



Este libro se ha diseñado como un curso de programación en Pascal en las versiones 5.0/5.5, 6.0 y 7.0 de Turbo Pascal. Puede ser utilizado con la nueva versión Borland Pascal 7.0 ejecutándose bajo el sistema operativo DOS. Esta nueva publicación busca conseguir dos objetivos complementarios: enseñar a programar con un estilo depurado y enseñar las nuevas técnicas de Turbo Pascal.

Con la tercera edición se ha ampliado considerablemente el contenido de las ediciones anteriores. Incluye alogaritmos, abstracción de datos, objetos, estructuras de datos, programación orientada a objetos y todas las bases de datos del lenguaje Pascal y programación en general.

Con la versión 7 de Turbo Pascal y Borland Pascal en el mercado, se ha considerado a este lenguaje como una potente herramienta de programación y de fácil aprendizaje, siendo de gran utilidad tanto en el mundo profesional como en el campo de la educación.

Editorial: McGraw-Hill
Nº de páginas: 1094
Nivel: medio

Autor: Luis Joyanes Aguilar
Idioma: castellano
Precio: 6.900

ENSAMBLADOR
Y SISTEMAS
OPERATIVOS

Funciones del 386 para Sistemas Operativos

Jorge Figueroa (erde@arrakis.es)

Como muchos de los lectores ya deben saber, los microprocesadores de la familia Intel x86, a partir de la generación x386 disponen de funciones avanzadas implementadas en hardware que están diseñadas expresamente para poder crear sistemas operativos seguros y avanzados, con capacidades de multitarea, multiusuario y de con capacidad de memoria virtual.

En este artículo se van a detallar todas las instrucciones ensamblador que poseen dichos microprocesadores para tales fines, y en qué consisten exactamente todas las utilidades que nos ofrecen.

Pese a que este área de la programación normalmente no es muy útil para el programador típico de aplicaciones, resulta interesante entender los conceptos explicados, puesto que siempre es positivo conocer como funcionan internamente los sistemas operativos actuales, como son Windows 95 o NT.

La gestión de memoria

El microprocesador dispone de dos métodos para la gestión de memoria, los cuales usados conjuntamente, son los que permiten a los sistemas operativos poder proteger la memoria y el uso de memoria virtual. A estos dos sistemas se les llama respectivamente segmentación y paginación. A continuación los vamos a explicar detalladamente.

Segmentación y Paginación

La CPU pone a disposición del Sistema Operativo dos sistemas de gestión de memoria que se pueden usar simultáneamente y que hacen entonces que el control de la memoria sea un proceso de dos etapas.

Los micros 386 poseen utilidades hardware diseñadas para sistemas operativos

El primer sistema de gestión de memoria, la llamada segmentación (nomenclatura herencia de la antigua arquitectura 80x86), se usa para poder gestionar la memoria que pertenece a cada aplicación y poder así proteger la memoria entre las diversas aplicaciones, haciendo por ejemplo que una supuesta

En este artículo se van a explicar todos los conceptos relacionados con las funciones hardware de que disponen los microprocesadores de la familia Intel a partir del modelo 386, las cuales, son las que permiten que actualmente haya sistemas operativos potentes, con capacidad multitarea y demás.

Figura 1. Esquema de como gestiona el microprocesador la memoria.

tarea A, no puede acceder a los bloques asignados a una tarea B y viceversa.

Como tener una forma de poder controlar cada byte de memoria de forma individual requeriría de una gran cantidad de información que lo hace en consecuencia inviable, el mapeado de memoria se hace por bloques, de forma que cuando una aplicación pide que se le asigne un bloque de memoria para uso propio, el sistema sólo necesita registrar tres valores para el mismo en una tabla especial que posee.

La gestión de memoria se realiza con un sistema de dos etapas

La llamada paginación de memoria se puede usar opcionalmente, ya que se puede activar y desactivar, y funciona siempre después de la segmentación como segunda etapa de la conversión de las direcciones de memoria usadas por las aplicaciones.

El esquema del funcionamiento de la gestión de memoria que realiza la C.P.U. internamente se puede ver de forma gráfica en la figura 1.

En el dibujo se puede ver ejemplificado como una aplicacion X, usa una instrucción que intenta acceder a una dirección **Selector: Desplazamiento.** Entonces, la C.P.U. comprueba si el segmento número *Selector* dentro de la tabla de Segmentación es válido y una vez hecho esto, tenemos que la dirección resultante es la dirección de memoria lineal, que en caso de que la paginación esté desactivada corresponde también con la dirección física de memoria, y en caso de que esta esté activada, debe entonces someterse al proceso de traducción según la tabla de paginación, con lo que finalmente habremos obtenido la dirección de memoria física.

La segmentación de memoria

La segmentación, como se ha dicho, divide la memoria en bloques, de forma que el sistema operativo sólo necesita de una pequeña tabla donde están los datos relativos a todos los segmentos que han sido definidos hasta el momento.

Por cada segmento que se define, en la tabla de segmentación se almacena solamente un valor de 64 bits que se conoce por el nombre de un *Descriptor*, y que no son más que tres datos definidores: 1-La dirección base (4 bytes), 2-el tamaño del bloque definido o límite (2 bytes) y 3-los atributos de dicho bloque (lectura/escritura, nivel de privilegio, etc...que son 2 bytes).

Para los mas curiosos, decir que en realidad hay dos tablas de descriptor de segmento, y ambas no son en realidad más que dos pequeños segmentos especiales que sólo se pueden alterar mediante el uso de instrucciones especiales ensamblador, que a su vez sólo son accesibles desde el sistema operativo.

Segmentación global y local

En realidad, el sistema divide la memoria en dos mitades. Usando en consecuencia dos tablas de descriptores simultáneas.

Estas son las llamadas <*Tabla Global de Descriptores*> y la <*Tabla Local de Descriptores*>, y que se conocen respectivamente por las siglas inglesas <*GDT*> y la <*LDT*>.

La primera, la GDT es fija, y siempre está presente, por lo que todas las aplicaciones que ejecute el sistema operativo la comparten. En cambio, la otra mitad de memoria virtual, la que se controla con la tabla LDT, cambia cada vez que cambiamos de tarea activa, por lo que tenemos que cada tarea posee una tabla local propia. De esta forma, cada aplicación posee la capacidad de tener segmentos definidos para uso exclusivo a la vez que puede compartir los segmentos de la tabla global.

Mapa de descriptores

Como se ha dicho anteriormente, cada descriptor de segmento definido, ya sea en la GDT o en la LDT, posee tan solo tres componentes, pero su estructura interna es algo más compleja, por lo que

se ha incluido un dibujo con el esquema interno de un descriptor en la figura número 2. Además de lo dicho, tenemos que en realidad hay tres tipos de descriptor, que son: Los *descriptores de memoria, los descriptores del sistema*, y *los descriptores de puertas,* los cuales difieren entre si solamente en la organización de los flags del elemento atributo del descriptor.

Esquema general de un descriptor de memoria o de sistema:

- Bits 0-15 (bytes 0 y 1): Límite del segmento (16 bits bajos).
- Bits 16-38 (bytes 2 a 4): Contienen los bits 0 a 23 de la Base del segmento.
- Bits 39-55 (bytes 5 y 6): Atributos, que varían según de cual de los dos tipos de descriptor se trate (mapa detallado más adelante).
- Bits 56-63 (byte 7): Aquí están contenidos los bits 24 a 31 de la base.

La memoria que usan las aplicaciones se llama memoria virtual

Contenido de los bytes 5 y 6 de atributo de un descriptor de memoria (bits 40 a 55):

- Bits 40-43: Tipo de descriptor (Ver posibles tipos en tabla 1)
- Bit 44: Flag DT1, distingue entre tipos de memoria.
- Bits 45-46: Flag DPL, contiene el nivel de privilegio del segmento de memoria. Hay 4 niveles posibles de protección.
- Bit 47: Flag P, del inglés <*Present*>. Cuando este flag está a 1, significa que el descriptor está activo y se puede usar para la traducción de direcciones. En caso contrario, si se intenta usar provocará un error (lo que se conoce como una excepción).
- Bits 48-51: Contiene los bits 16 a 19 del Límite.
- Bit 52: Flag AVL. Proviene del inglés <*Avaiable-to-software*>. Este bit no lo usa el 386, por lo que está a disposición del programador.
- Bit 53: Siempre está a 0.
- Bit 54: Posee el flag D. Este bit debe poseer el valor 1 siempre en un 386, mientras que en los 286 estaba a 0.
- Bit 55: Posee el flag G. Con este bit se define la <*Granularidad*> del límite de segmento. Cuando este bit es 1, la granularidad es de una página (4 Kbytes), si G=0 la granularidad es de byte.

Contenido de los bytes 5 y 6 de atributo de un descriptor de sistema:

- Bits 40-43: Tipo de descriptor de sistema (Ver los posibles tipos dentro de la tabla 2).
- Bit 44: Flag DT0. Este flag debe de estar a 0.
- Bits 45-46: Flag DPL, contiene el nivel de privilegio del segmento de sistema. Hay 4 niveles posibles de protección.
- Bit 47: Flag P, del inglés <*Present*>. Cuando este flag está a 1, significa que el descriptor está activo.
- Bits 48-51: Contiene los bits 16 a 19 del Límite.
- Bit 52: Flag AVL. <*Avaiable-to-software*>.
- Bit 53: Siempre está a 0.
- Bit 54: Este bit, que en el tipo de memoria era el Flag D, aquí no se considera útil.
- Bit 55: Posee el flag G. Con este bit se define también la <*Granularidad*> del límite de segmento. Cuando este bit es 1, la granularidad es de una página (4 Kbytes), si G=0 la granularidad es de byte.

Esquema general de un descriptor de puerta:

TABLA 1. Tipos de descriptor de memoria según el flag TIPO.

0: Es de sólo lectura.
1: Sólo lectura, accedido.
2: Lectura y escritura.
3: Lectura y escritura, accedido.
4: Sólo lectura con el límite expandible hacia abajo.
5: Sólo lectura con el límite expandible hacia abajo, accedido.
6: Lectura y escritura con el límite expandible hacia abajo.
7: Lectura y escritura con el límite expandible hacia abajo, accedido.
8: Sólo de ejecución.
9: Sólo de ejecución accedido.
A: Lectura y escritura.
B: Lectura y escritura, accedido.
C: Sólo de ejecución, de conformidad.
D: Sólo de ejecución, de conformidad accedido.
E: Ejecución y lectura, de conformidad.
F: Ejecución y lectura, de conformidad y accedido.

TABLA 2. Tipos de descriptor de sistema o de puertas según el flag TIPO.

0: Indefinido.
1: Disponible TSS 286.
2: LDT.
3: Ocupado TSS 286.
4: Puerta de llamada 286.
5: Puerta de tarea.
6: Puerta de interrupción 286.
7: Puerta de trap 286.
8: Indefinido.
9: Disponible TSS 386.
A: Indefinido.
B: Ocupado TSS 386.
C: Puerta de llamada 386.
D: Indefinido.
E: Puerta de interrupción 386.
F: Puerta de trap 386.

- Bits 0-15 (bytes 0 y 1): Desplazamiento (16 bits bajos).
- Bits 16-31 (bytes 2 y 3): Selector.
- Bits 32-47 (bytes 4 y 5): Atributos (mapa detallado a continuación).
- Bits 48-63 (bytes 6 y 7): Desplazamiento (bits 16 a 31).

Cuando la memoria virtual pasa por la segmentación, se convierte en memoria lineal

A continuación mostramos el contenido de los bytes 4 y 5 de atributo de un descriptor de sistema:

- Bits 32-36: Cuenta-Doblepalabra.
- Bits 37-39. Estos bits siempre se encuentran a 0.
- Bits 40-43: Tipo de descriptor (Los tipos posibles son los mismos que para el tipo de sistema, vistos en la tabla 2).
- Bit 44: Flag DT0.
- Bits 45-46: Flag DPL. (Nivel de privilegio).
- Bit 47: Bit de Flag P (*Present*).

La paginación de memoria

Como se ha dicho, también disponemos de la llamada paginación de memoria, que cuando está activa, es la segunda etapa en la conversión de direcciones virtuales a físicas, ya que coge las direcciones lineales que se hayan obtenido con la segmentación y las convierte en físicas.

Para activar y desactivar la paginación, disponemos de un bit específico para ello llamado PG (*PaGing*), que no es más que una bandera de estado que esta contenida en el registro especial CR0, el cual, por supuesto, sólo se puede alterar desde un nivel de privilegio 0, o sea, desde el sistema operativo.

La diferencia principal entre la paginación frente a la segmentación es que mientras en la segunda los bloques de memoria que se manejan son de tamaño variable, en la paginación los bloques de memoria que se gestionan son de tamaño un fijo de 4 Kbytes siempre. A cada uno de estos bloques se le llama página, y de ahí viene el nombre de Paginación al sistema de manejo de dichos bloques de la C.P.U.

El funcionamiento de la paginación es muy fácil de entender. El sistema posee una tabla donde cada página lineal de memoria posee un equivalente en las páginas físicas en que se ha dividido la memoria.

Por defecto, cada página lineal posee asociada la página física que ocupa su misma dirección, la misma ubicación por así decirlo, pero simplemente cambiando las equivalencias de las tablas, podemos hacer , como es fácil deducir, que haya páginas físicas que nunca sean accesibles cuando queramos, con lo cual se consigue la llamada protección de memoria.

La tabla que almacena estas equivalencias se llama <*Tabla de Paginación*>.

Problemas teóricos de la paginación

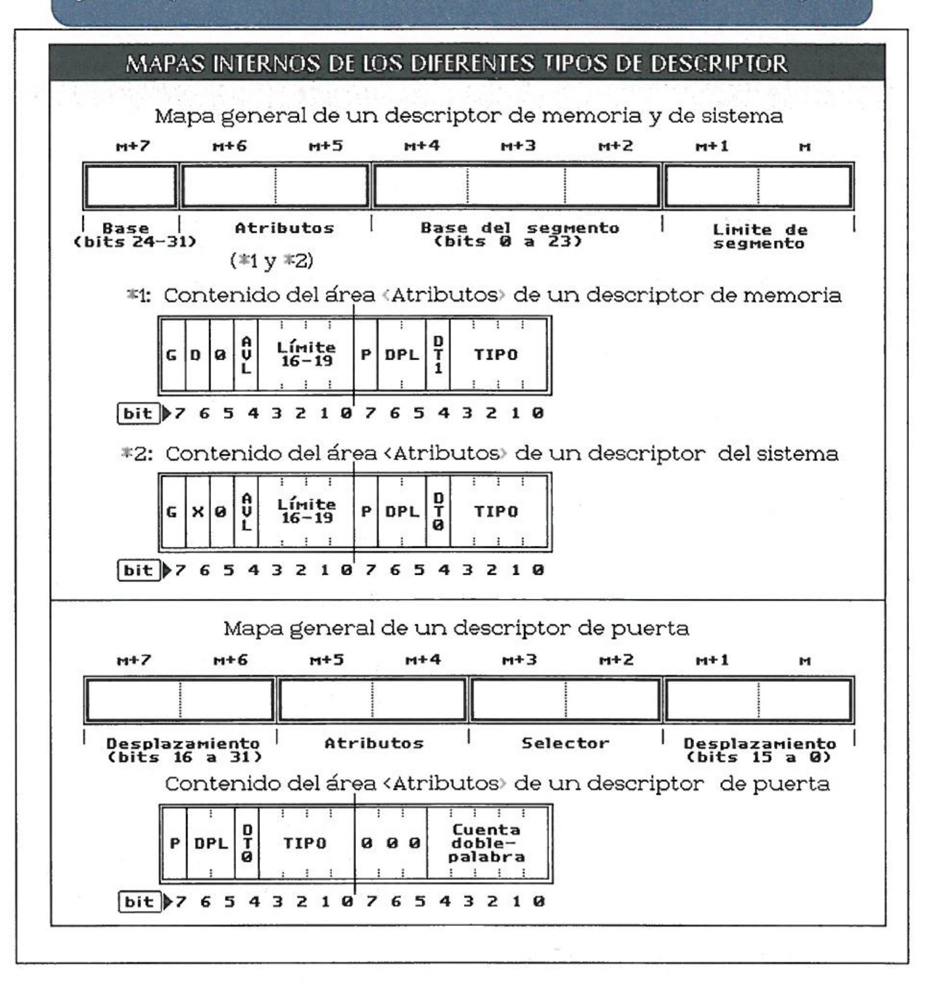
Si hacemos unos cuantos números, veremos que para crear una tabla con tan sólo 1 elemento de 32 bits por cada página de la memoria, y teniendo en cuenta que estos microprocesadores pueden direccionar 4 Gbytes, da como resultado que necesitaríamos una tabla contigua de 4 Mbytes de tamaño para poder albergarlas todas. Ello es mucho, y más si tenemos en cuenta que en casi todos los casos el sistema operativo estará instalado en un ordenador que posee 8, 16 o 32 Mbytes de R.A.M., lo cual dejaría a la máquina casi inutilizada en cuanto a recursos se refiere.

Si la paginación está deshabilitada, la memoria lineal corresponde con la física

Para evitar este problema, los diseñadores del micro, crearon un sistema de paginación de dos niveles. Teniendo así que en vez de tener una sola tabla contigua con todas las páginas definidas, disponemos de una tabla que ocupa sólo 4 Kbytes, 1 página física, y que se usa como

índice puntero a 1024 tablas de segundo nivel, que existirán sólo en aquellos casos en que sean necesarias (en que la memoria a que correspondan exista), ahorrando de esta forma mucha memoria y permitiendo a la vez, que no sea obligado tener todas las definiciones almacenadas de forma contigua.

Figura 2. Esquema de la estructura interna de los tres tipos de descriptor de segmento.



Paginación global y local

En el método de paginación, también existen tablas tanto globales y locales. Para hacerlo, se define una parte del espacio lineal de direcciones como global. Entonces tenemos que las tablas de paginación que definamos en este área, serán globales, porque siempre se convertirán en direcciones físicas, en cambio, las páginas que definamos en el restante espacio virtual de memoria, serán locales.

Estructura de las Tablas de paginación

Tanto las páginas definidas en las tablas globales como las definidas en las locales, poseen la misma estructura de definición, y que es la de la figura número 3, donde se puede ver que cada página se define como un elemento de 32 bits.

A continuación de explica cada elemento:

- Bit 0: Es el Indicador P (de *Presente*) e indica si dicha entrada es válida o no, queriendo decir por válida si está disponible para usarse en la traducción de direcciones.

- Bit 1: Flag R/W (*Read/Write*). Si este bit está a 1, significa que dicha página se puede leer, escribir y ejecutar. Si está a 0, La página se puede leer o ejecutar, pero nunca escribir.

- Bit 2: Aquí está el flag U/S (de *Usuario/Supervisor*). Si este bit está a 1, la página es accesible a los programas que se ejecuten en cualquier nivel de privilegio (0-3). En cambio, si el bit está a 0, la página sólo es accesible desde niveles de protección entre 0 y 2, y no desde el nivel 3.

- Bits 3-4: Estos dos bits siempre están a 0, no son usados por la CPU.

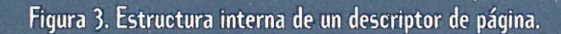
- Bit 5: Es el Flag A (de *Accedido*). Este bit se pone a 1 cada vez que dicha entrada se use para realizar una conversión de memoria, o sea, cada vez que la página física correspondiente se use.

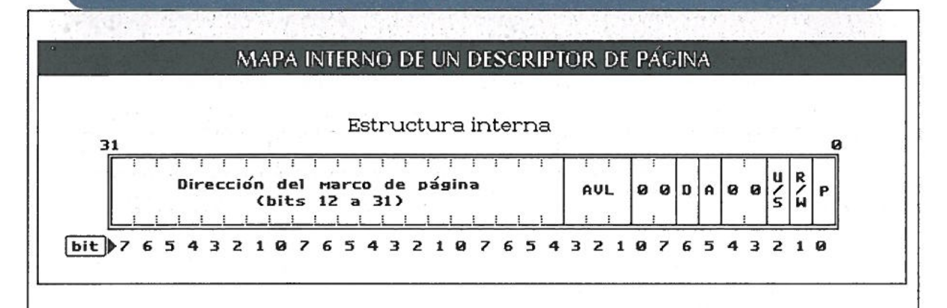
- Bit 6: Flag D (del inglés <*Dirty*>, sucio). Este bit se pone a 1 siempre que se haga alguna operación de escritura de memoria en la página física a la que apunta.

En una tabla de segmentación, podemos encontrar tres clases de descriptores

- Bits 7-8: No se usan, siempre están a 0.

- Bits 9 a 11: Son 3 bits del campo llamado AVL, que proviene del ingles <Avaiable>, que significa *disponible*.





La C.P.U. nunca los modificará ni usará para nada.

- Bits 12 a 31: Son los bits de la parte superior del Offset físico donde empieza la página a la que apunta en la memoria física.

Instrucciones de sistema operativo

Ahora que ya se ha explicado la forma en que gestiona el microprocesador la memoria, se van a detallar un poco las instrucciones ensamblador que están relacionadas entre otros apartados con dicha gestión y que son las llamadas *comúnmente instrucciones de sistema operativo*. La mayoría de ellas, solamente pueden ejecutarse desde un nivel de privilegio 0 (de los 4 que se han dicho que había), que corresponde al nivel del S.O., lo que significa que no pueden ejecutarse desde las aplicaciones normales.

Este grupo de instrucciones son un total de 18, de las cuales, se van a explicar solamente las que tienen relación con lo explicado en este artículo, que son unas 12:

- ARPL *op1,op2*: Esta instrucción comprueba el nivel de privilegio (RPL) solicitado de Op1 comparándolo con el de Op2. Si el RPL de OP1 es menor al de OP2, el flag ZF se pone a 1. Esto se usa para comparar privilegios de selectores y permite comprobar rápidamente si se está intentando usar un selector con un privilegio mayor del que tiene definido.

- LAR *op1,op2*: Esta instrucción, carga en el primer operando los atributos del descriptor de segmento indicado en el operador 2.

- LGDT *op1*: Esta instrucción, que proviene del inglés <*Load Global Descriptor Table*>. Carga la tabla de descriptores indicada en Op1, que es un puntero completo de 48 bits (base:límite).

Hay 12 instrucciones destinadas al manejo de las utilidades de gestión de memoria

- LIDT *op1*: Proviene del inglés <*Load Interrupt Descriptor Table*>. Esta instrucción es idéntica a la anterior pero sirve para cargar la tabla de descriptores de interrupción, que es un apartado que no se ha explicado por salirse un poco del tema del artículo.

- LLDT *op1*: Proviene del inglés <*Load Local Descriptor Table*>. Esta instrucción es idéntica a las LGDT, pero sirve para cargar la tabla de descriptores local y no la global.

- LSL *op1,op2*: Esta instrucción, carga en el primer operando el límite del descriptor de segmento apuntado por el selector indicado en el operador 2.

- LTR *op1*: Carga en el *registro de tarea* del selector contenido en el operador 1.

- SGDT *op1*: Esta instrucción carga el contenido del registro de la tabla de descriptores global de la posición de memoria indicada en el operador 1. En otras palabras, almacena en Op1 la base y longitud donde está contenida la GDT actual.

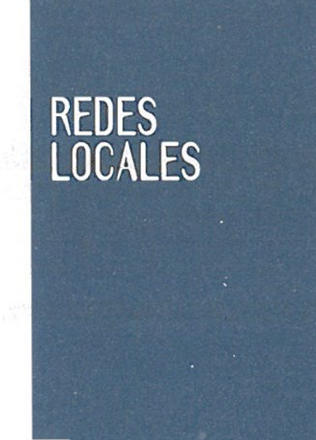
Las instrucciones del S.O. sólo pueden ejecutarse desde un nivel de privilegio 0

- SIDT *op1*: Esta instrucción realiza la misma función que la anterior pero con el registro de la tabla de descriptores de interrupción.

- SLDT *op1*: Esta instrucción realiza la misma función que SGDT pero con el registro de la tabla de descriptores local en vez de la global.

- VERR *op1*: Esta instrucción detecta si el segmento correspondiente al selector contenido en el operador 1 permite la lectura. Si la comprobación sale positiva, el flag ZF se pone a 1.

- VERW *op1*: Esta instrucción detecta si el segmento correspondiente al selector contenido en el operador 1 permite escritura. Si la comprobación sale positiva, el flag ZF se pone a 1.



Cactus 3.0 de Information Builders

Enrique Díaz (enriqued@jet.es)

Hace algunos años, cuando programar en Windows era algo reservado a los *gurús* del C o Pascal, cayó en mis manos un programa de Borland llamado ObjectVision. La filosofía del programa era *enganchar* bases de datos a las ya omnipresentes ventanas de Windows. Cactus, algunos años más tarde, cumple la misma función, pero esta vez enganchamos las bases de datos a las ya omnipresentes páginas Web.

Cactus se nos presenta como un RAD (*Rapid Application Development,* desarrollo rápido de aplicaciones) que utiliza su propio lenguaje de programación, el Maintain, que viene a ser algo así como una mezcla de COBOL y SQL.

Cactus permite crear aplicaciones cliente-servidor, ya sea en la Web o fuera de ella, de modo que es válido para crear aplicaciones para el comercio electrónico. Como todas las aplicaciones de desarrollo visual, Cactus está muy orientado al objeto y ofrece facilidades para la reutilización de componentes. Además soporta Java, JavaScript, CGI's y NSAPI, así como controles VBX, OLE y ActiveX. Podemos Integrar todos estos elementos en nuestras aplicaciones, e incluso distribuir el desarrollo realizando partes con herramientas como Visual Basic. Así pues, para crear aplicaciones profesionales en la Web, no necesitamos ser unos genios del HTML, ni saber mucho sobre Internet. Si sabemos programar en cualquier tipo de lenguaje, sabremos realizar una aplicación con Cactus.

La instalación

La instalación de Cactus es absolutamente sencilla: el proceso de instalación sólo nos pedirá confirmación de los nombres de los directorios que albergarán el programa. La versión que probamos se integró perfectamente con Windows NT 4.0. Incluso detectó el *Internet Information Server* que estaba instalado —que por cierto en ese momento no estaba activo— y pidió permiso para configurarlo para trabajar con Cactus. Eso es todo lo que tuvimos que hacer y no quedó nada más que configurar. El programa queda listo para usarse.

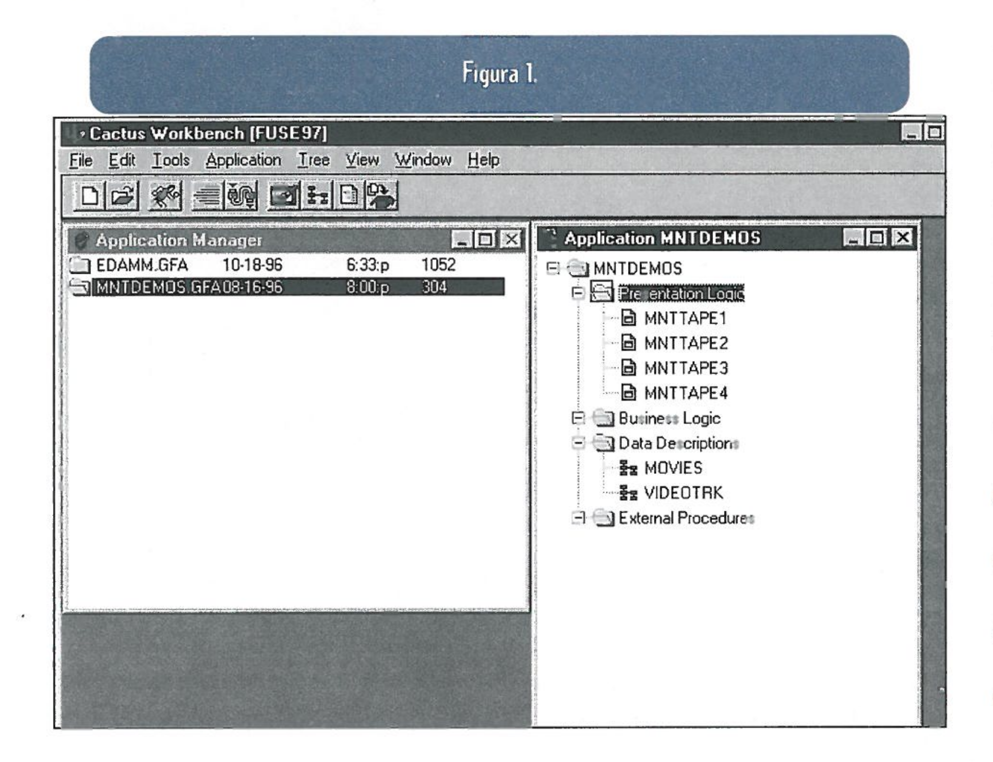
Independientemente de la versión para Windows NT, se ofrece también para las plataformas de HP-UX, Sun, IBM RISC, NCR e IBM/MVS; y podremos atacar las bases de datos más comunes, como Microsoft SQL Server, DB2, Btrieve, Teradata...

El Workbench

El Workbench (mesa de trabajo) es la primera pantalla que se nos aparece cuando iniciamos Cactus (figura 1). Desde ella tenemos acceso a todas las herramientas que necesitaremos para crear los componentes de la aplicación que realicemos. Estos componentes están categorizados por funcionalidades: acceso a datos (*Data*

Analizamos en esta ocasión Cactus 3.0, un gestor de información diseñado para ser todo un especialista en su función: crear una serie de aplicaciones cliente-servidor de fácil transporte a la Web.

Figura 1.



access), lógica de funcionamiento (*Business logic*), y presentación (*Presentation layer*).

Aunque el Workbench está bien estructurado, resulta difícil trabajar con él con resoluciones de 640x480 —la más habitual en los servidores—, ya que entonces no caben todos los elementos en la pantalla y hay que andar moviendo las barras de desplazamiento continuamente, lo cual llega a ser bastante engorroso. Así pues, lo mejor será trabajar con la resolución de 800x600.

Al abrir el Workbench se nos muestra por defecto la ventana del *Application Manager,* que nos mostrará los componentes de nuestra aplicación separados por las funcionalidades ya comentadas: *Presentation logic, Business logic* y *Data descriptions.*

Crear una aplicación

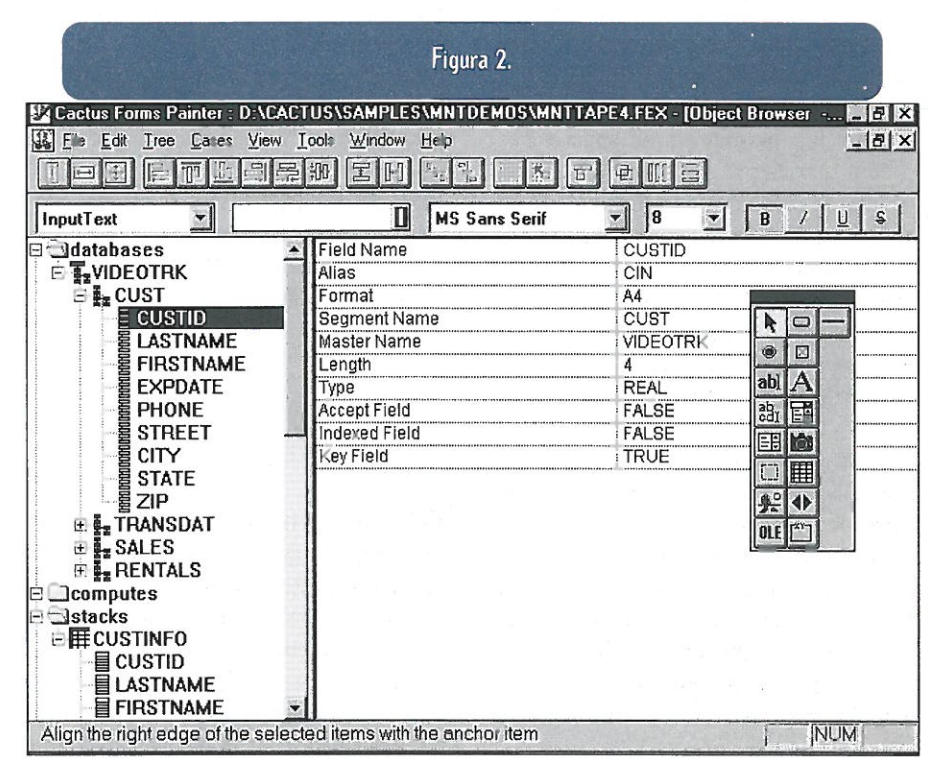
Como el movimiento se demuestra andando, vamos a crear una pequeña aplicación como ejemplo de las capacidades y el uso de Cactus.

En primer lugar seleccionamos la ventana *Application manager* del *Workbench,* y pulsamos *New* bajo el menú *Application.* Con esto creamos un repositorio para los objetos de la nueva aplicación. Aunque pueda parecer que las aplicaciones se corresponden con directorios (de hecho el icono empleado para la aplicación es el mismo que usa Windows para los directorios), lo que en realidad estamos haciendo es crear un archivo con la extensión .GFA que contiene el nombre y la localización de la aplicación, así como de todos los componentes que forman parte de ella.

Cactus 3.0 está muy orientado al objeto y ofrece facilidades para la reutilización de componentes

Si efectuamos una doble pulsación sobre la recién creada aplicación veremos una nueva ventana, con la misma filosofía de árbol de directorios, en la que aparecen los elementos de la aplicación separados por sus categorías. Crearemos en primer lugar un Presentation logic, que es lo que en otros lenguajes visuales llaman formu-

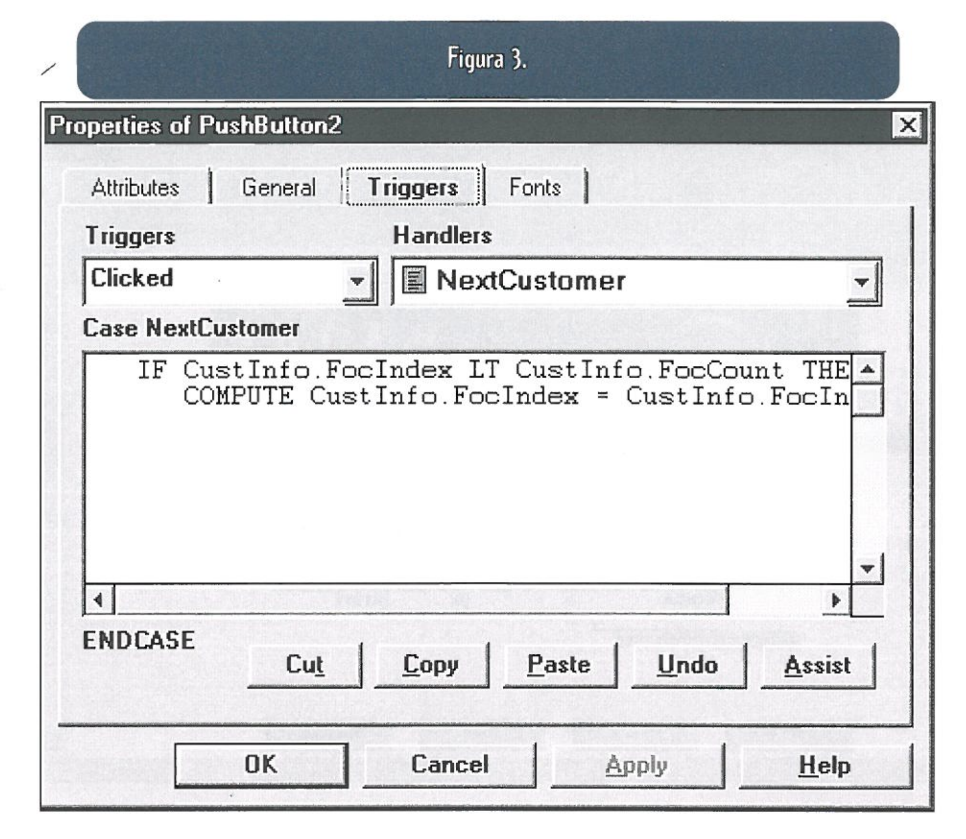
Figura 2.



larios o ventanas; aunque en este caso el concepto es algo diferente, ya que un sólo *Presentation logic* puede contener varios *Forms* o formularios.

Cactus está tan orientado al dato que, tras suministrarle el nombre del nuevo *Presentation logic*, lo primero que hace es preguntarnos por la fuente de datos para la aplicación, de modo que lo más cómodo es establecer las fuentes de datos en el *Data description* antes de crear el nuevo formulario.

Figura 3.



El Form Painter

El *Form Painter* es la herramienta de diseño visual de Cactus; y a través de ella podemos acceder y crear otros elementos de la aplicación: por ejemplo un *Business logic*. Su aspecto, para los entornos que se estilan hoy en día, es un tanto *retro*, pero eso no quiere decir que sus métodos lo sean.

Para añadir objetos y controles disponemos de una barra de herramientas; bastante cortita, dicho sea de paso. De cualquier modo, es más que suficiente y tiene a su favor el hecho de que podemos añadirle cualquier control VBX que tengamos a mano. Aunque si nuestra aplicación se va a ejecutar en la Web deberemos quitar los VBX para que funcione correctamente. Veamos cuales son las herramientas más significativas:

- *Push button*. Botones que el usuario puede pulsar y que desencadenarán un procedimiento creado por nosotros.
- *Radio button group*. Botones tipo radio que el usuario podrá seleccionar. Van enlazados a la base de datos y su número y el texto que aparecerá se determina dinámicamente en tiempo de ejecución dependiendo del número de filas la fuente de datos. Se generará un ítem por cada fila.
- *Check box*. Casilla de verificación a la que se le asigna automáticamente una variable que puede tomar los valores 0 y 1. También podemos desencadenar un procedimiento directamente cuando se selecciona o deselecciona.

Para poner campos enlazados a datos en un formulario sólo hay que arrastrar y soltar desde el Object browser

- *Single line entry field*. Caja de texto. Es una caja de texto en la que podemos mostrar información o hacer que el usuario la introduzca. Se puede enlazar a una fuente de datos. También disponemos del *Multi-line entry field* si deseamos tener varias líneas de texto.
- *Text Box*. Es lo que en otros entornos llaman *Labels*. Sirve para mostrar textos estáticos no editables.
- *Combo box* y *List Box*. Listas desplegables. Están enlazados a datos y se puede especificar que el contenido de la lista se establezca dinámicamente en tiempo de ejecución. Los *Combo Box, además*, pueden mostrar una lista de valores aceptables para un campo, según los valores establecidos en la descripción del archivo.
- *Hyper region*. Establece una zona sensible en un formulario; son invisibles y, por tanto, quedan recubiertos por otros objetos del formulario; podría ser un ejemplo una imagen. Por lo demás, funcionan como un botón completamente normal. Tienen la pega de que también son invisibles en tiempo de diseño, con lo cual es mejor poner primero la imagen -o lo que sea- y después establecer el área de la zona sensible.
- *Picture box* y *Picture button*. El *Picture box* nos permite poner una imagen en el formulario por motivos meramente decorativos, y el *Picture button* hace que esa imagen funcione como un botón.

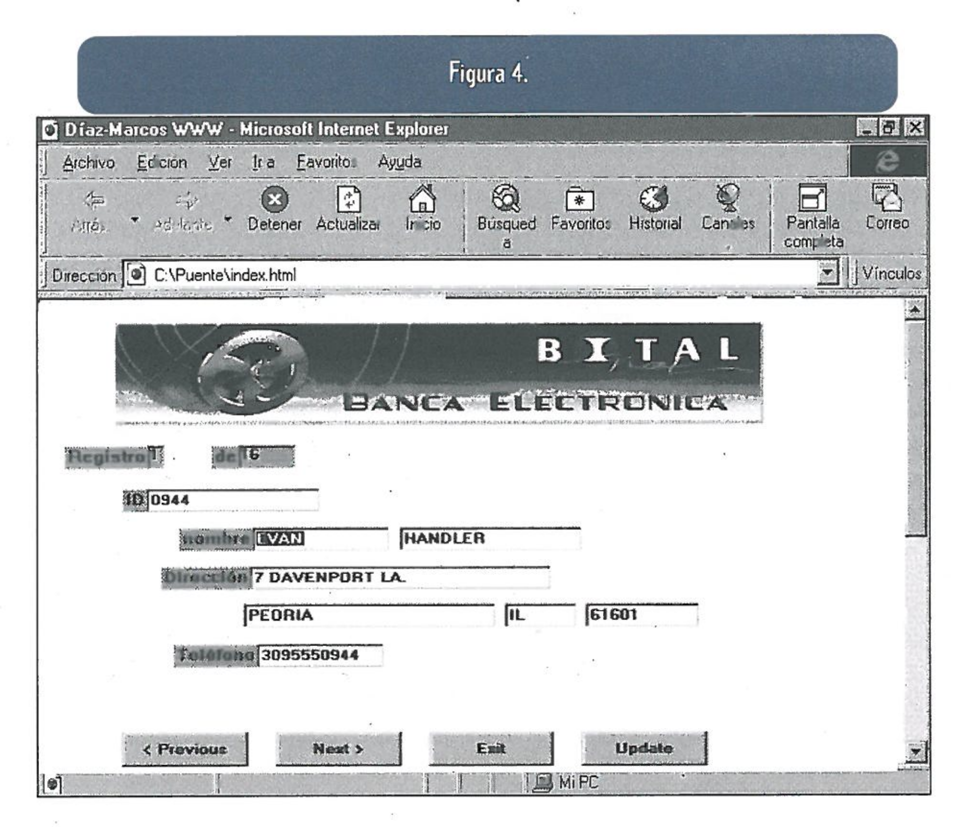


Figura 4.

- *OLE object*. Nos permite ver o ejecutar datos de diferentes aplicaciones como Word, Excel, clip arts o sonidos.
- *Grid y Browser.* Nos permiten ver editar e introducir datos en la base de datos. Con el Browser lo haremos de uno en uno, mientras que con el Grid veremos varios registros a la vez.

Como vemos, todo está muy orientado al dato de modo que si queremos algún otro tipo de funcionalidades tendremos que agregar controles VBX que deberán ser distribuidos con la aplicación. Junto con los VBX deberemos también proveer los siguientes componentes para que la aplicación funcione correctamente:

- Objetos OLE. Al igual que con los VBX, deberemos tener permiso para su distribución.
- Descripciones de archivo. Por cada archivo de base de datos utilizado en la aplicación deberemos suministrar un archivo con la descripción de los datos. Esto lo podemos realizar automáticamente eligiendo *New Data description* bajo el menú *File*.
- Formularios. Los formularios se almacenan en archivos con la extensión WFM. Por cierto, es mejor no editar estos ficheros como texto para introducir modificaciones, ya que se perderán los cambios al abrirlos de nuevo con el *Form painter*.
- El archivo de procedimiento de cactus. Este archivo tiene la extensión FEX y almacena el procedimiento principal de nuestra aplicación. Esto es, el *Business logic*.

Podemos cambiar una aplicación para que trabaje en la Web con sólo pulsar un botón

Un detalle francamente bueno de esta aplicación es la manera de poner campos enlazados a bases de datos: simplemente con arrastrarlos desde el Object Browser (figura 2) y soltarlos en formulario tendremos los campos puestos, dispuestos y con sus correspondientes etiquetas descriptivas. Un detalle que los programadores agradecemos enormemente por su comodidad.

El Object browser nos permite acceder de manera estructurada a todos los objetos y partes de nuestra aplicación. Desde aquí podemos editar y crear procedimientos, propiedades, bases de datos, etc. Resulta muy útil para modificar aplicaciones o dar los últimos retoques a una casi acabada. Así, para poner los campos de la base de datos en el formulario, solo hay que seleccionar los campos que nos interesan, bajo el epígrafe *Databases,* y arrastrar y soltar sobre el formulario. Sólo nos restará alinear correctamente los campos y crear los procedimientos, si es necesario.

Propiedades y eventos

Los objetos que ubiquemos en el formulario tienen unas propiedades y responden a unos eventos (figura 3). El modo de manejarlo no es muy diferente, en su concepto, a otros lenguajes visuales. Por una parte podemos establecer las propiedades de los objetos, como su color, situación, sus enlaces con las bases de datos, etc. Por otra parte definiremos el código a ejecutar dependiendo del evento recibido.

Para codificar las acciones a ejecutar (el *Business logic*) disponemos del lenguaje Maintain (listado 1). Este lenguaje, como ya hemos dicho, se asemeja bastante a una mezcla de SQL y COBOL. El lenguaje Maintain nos permitirá establecer el control de flujo del programa, así que una de las primeras cosas que debemos hacer es darle un formulario a ejecutar. A partir de ahí los programas se ejecutan como una gran estructura *case*. Es decir, le damos un formulario para ejecutar y bifurcamos la ejecución hacia puntos del código dependiendo de los eventos que reciba el formulario. Por supuesto, una de las opciones posibles es ejecutar otro formulario.

Para facilitarnos ésto disponemos también de un asistente para crear código. Para iniciarlo seleccionaremos *New Case* bajo el menú *Cases*. También podemos crearlo desde el *Object browser*, seleccionando la carpeta Cases y sacando el menú contextual pinchando con el botón derecho del ratón.

El programa viene con un completo manual de referencia de este lenguaje, que no tiene mucho misterio si sabemos algo de SQL. Es recomendable darle un repaso a las variables de entorno del lenguaje, ya que son muy útiles y si no las utilizamos perderemos algunas funcionalidades importantes.

Listado 1. Ejemplo de código Maintain.

```
MAINTAIN FILE videotrk

PERFORM CustomerFetch
PERFORM Winform_ShowCust

CASE CustomerFetch
FOR ALL NEXT CustID INTO CustInfo
  WHERE ExpDate GE 920601 AND ExpDate LE 920621
endcase

case CustomerUpdate
FOR ALL UPDATE LastName FirstName Street City State Zip
  Phone FROM CustInfo
endcase

CASE NextCustomer
  IF CustInfo.FocIndex LT CustInfo.FocCount THEN
    COMPUTE CustInfo.FocIndex = CustInfo.FocIndex +1;
endcase

CASE PreviousCustomer
  IF CustInfo.FocIndex GT 1 THEN
    COMPUTE CustInfo.FocIndex = CustInfo.FocIndex - 1;
endcase
case Winform_ShowCust
  Winform show ShowCust
endcase

END
```

Para codificar las acciones a ejecutar disponemos del lenguaje Maintain

Otra posibilidad muy interesante es dividir nuestra aplicación de modo que se reparta la carga de trabajo entre varios servidores. De este modo podemos hacer que un servidor se encargue de validar las conexiones y presentar un formulario. Si el usuario pulsase un botón para agregar un registro, podemos hacer que esa parte del código se ejecute en otro servidor distinto, o si pulsa un botón para editar los datos existentes, que se ejecute en un tercero. Esta característica nos permite implementar el modo de trabajo típico en las aplicaciones cliente-servidor.

Nuestra aplicación en la Web

Para crear una aplicación para la Web, sólo tendremos que poner el *Form Painter* en modo Web. El modo de trabajar es el mismo, pero la barra de herramientas nos cambia un poco. Si tenemos una aplicación ya hecha y deseamos convertirla en una aplicación Web, tan sólo debemos abrirla normalmente con el *Form painter* y seguidamente ponerlo en modo Web y salvar.

Debemos tener en cuenta, eso sí, que no todos los objetos son transportables a la Web, por lo que antes deberemos quitar los objetos exclusivos de aplicaciones Windows, como son los VBX, los grids, menús, Hiperregiones y objetos OLE. A cambio podremos incluir applets de Java, JavaScript, CGI's...

Una vez hecho esto, cuando saquemos las propiedades del objeto tendremos una pestaña llamada HTML en la ventana de propiedades. Es aquí donde podremos agregar extensiones HTML a los objetos de la aplicación. Por ejemplo podemos establecer un fondo para el formulario.

Por lo demás, podemos trabajar como con cualquier documento HTML, es decir, podemos añadir enlaces, imágenes, applets, etc.

Finalmente, podemos ejecutar nuestras aplicaciones para la Web por el sencillo método de llamar a un CGI y pasarle el nombre de la aplicación y una identificación de usuario. Por ejemplo:

<p>Pulsa Aquí</p>. De este modo tendremos nuestra aplicación trabajando en la Web

JUEGOS

Ajedrez
El uso del tiempo

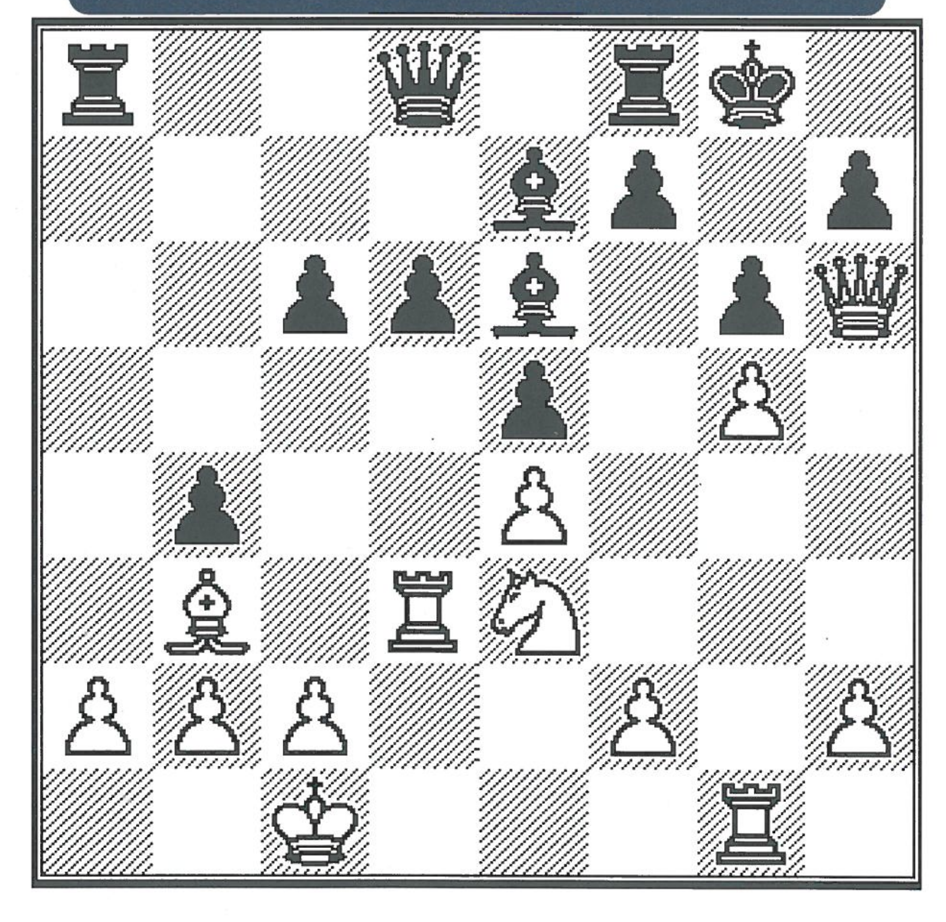
Eugenio Castillo Jiménez (ejimenez@lince.lander.es)

Las computadoras han sido capaces de mostrarnos con éxito las soluciones a numerosos juegos, sin embargo, el juego del ajedrez continúa resistiéndose a ser resuelto. Aquella herramienta que nos dé su secreto ha de ser algo más elegante que un simple analizador de posibilidades.

Por ahora la calidad de la respuesta de un programa de ajedrez depende en gran medida del tiempo que disponga para responder a la posición en que se encuentre, de hecho la mayor parte de los programas comerciales serían capaces de encontrar la mejor jugada posible si dispusiéramos de unos cuantos miles de años de paciencia para esperarlos.

Para "regular" dicha espera los programas actuales poseen un sistema de control de tiempo seleccionable por el usuario. Las modalidades a escoger pueden ser:

Figura 1. Mueven negras.



1º Control de n jugadas en x minutos. Cada n jugadas se emplean x minutos como máximo. Es la modalidad utilizada en competiciones de torneo serias, 60 jugadas en 2 horas es lo más común. En competiciones de programas se hacen 30 jugadas en una hora para el primer control, y luego 40 jugadas por hora para los siguientes.

2º Tiempo medio por jugada. Cada jugada se realiza en n segundos de media, 15 segundos es lo normal aunque la respuesta en algunos casos puede ser más larga.

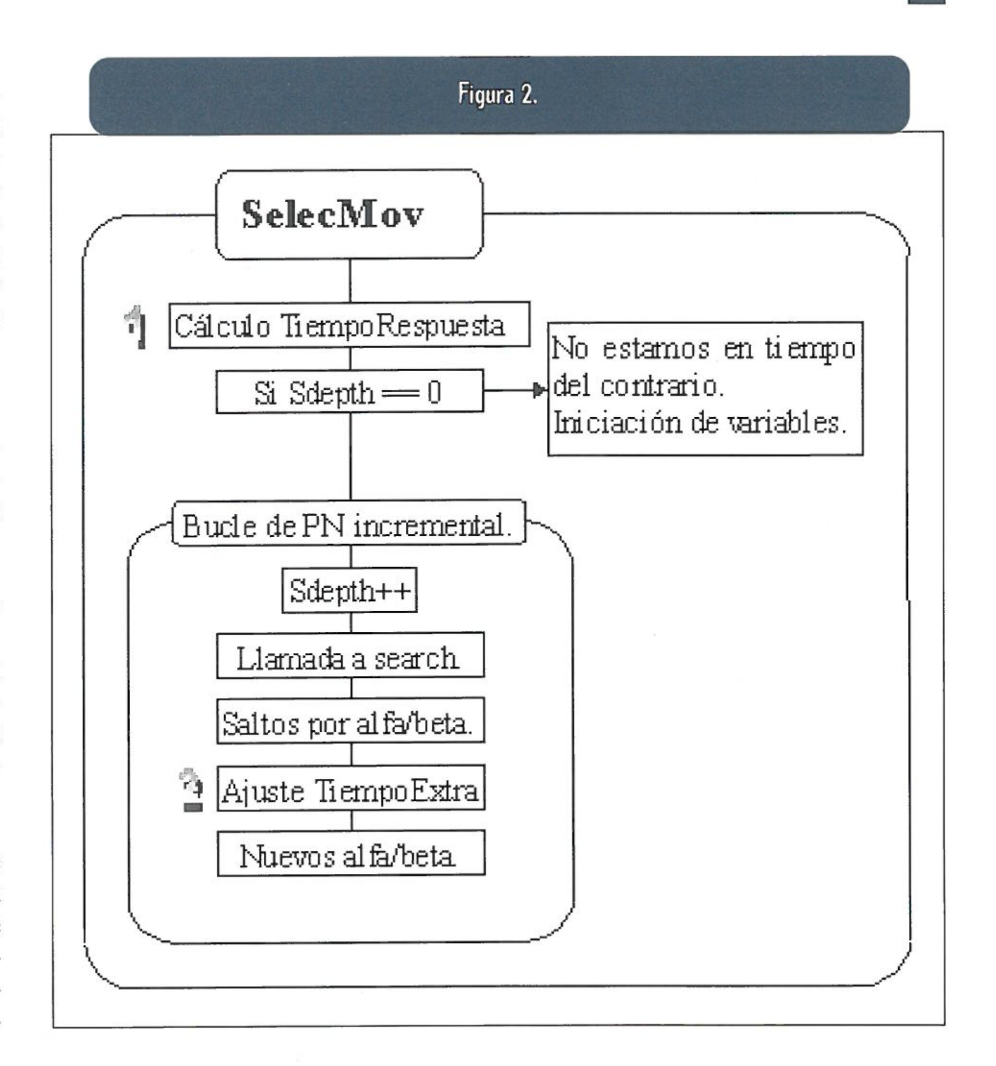
3º Tiempo total por partida (tiempo finish). Se conceden n minutos para toda la partida, las partidas rápidas y semirápidas se hacen con este tipo de control de tiempo. Los 10 y 30 minutos suelen ser los más aceptados.

Puesto que un programa requiere de un tiempo de proceso para generar una respuesta y dado que este tiempo se ve limitado por la modalidad de control elegida, es obvio que hemos de desarrollar criterios para usarlo con el máximo de eficacia posible.

El modo inicial más simple pudiera ser dividir el tiempo restante entre el número de jugadas que queden para el control. Si estamos en un sistema de tiempo finish se concede un margen constante de x jugadas previstas para el final (por ejemplo 30), independientemente del número de las ya realizadas. Esta solución es válida pero obviamente simple y en mucho casos insuficiente. Para obtener el máximo rendimiento del tiempo concedido hemos de intentar cumplir las siguientes máximas:

1º Consumir mayor cantidad de tiempo en las jugadas conflictivas.
2º Usar menos tiempo en las jugadas obvias.
3º Usar el tiempo de reflexión del contrario.
4º Consumir menos tiempo que el rival.

Figura 2.



Es evidente que los dos primeros puntos son los más difíciles de satisfacer puesto que hemos de dar criterios que discriminen entre jugadas obvias y jugadas conflictivas.

Veamos en primer lugar qué puede ser una **jugada conflictiva** para un programa. El caso más inmediato sería el de una variante que en profundidad nominal 5 fue tomada como buena y que al ser reanalizada a profundidad 6 es considerada como mala o perdedora, esto significa que ha caído por debajo del valor alfa inicial previsto (recordemos que el alfa inicial es un valor hipotético de referencia), lo cual hace preciso una revisión de la situación con un alfa inferior, algo que implica un consumo de nodos en algunos casos muy superior al normal. Si nos encontramos al límite o fuera del tiempo asignado es posible que el programa no sea capaz de encontrar una jugada viable.

La posición del diagrama 1 es para muchos programas un claro ejemplo de situación conflictiva como la anteriormente descrita, las blancas mediante Cf5 amenazan mate con Dg7, si las negras capturan el caballo de f5 la posición se torna desesperada porque ahora tras Th3 de las blancas, la amenaza de mate viene mediante Dh7. En su análisis inicial las negras ven 1. ... Axb3; 2. cxb3; Txa2 ganando un peón y Cf5 (la jugada asesina) es desechada porque las blancas pierden material sacrificando el caballo. No es hasta profundidad nominal de 6 a 8 (hay programas que debido al uso del turno nulo no identifican la variante ni a profundidad 11) que no se ve la validez de la combinación blanca.

Como podemos observar en esta posición se da la circunstancia de la caida de un alfa (valor inicial supuesto) con la jugada Axb3 negra como falsa variante principal. Si

Listado 1. Punto 1 de la figura 2.

```
 Inicialización del tiempo al principio de la búsqueda/iteracciones
*/

struct TiempoControl {long reloj;            // cantidad de tiempo restante en segundos
        short movimientos; //contador hacia atrás de movimientos respecto al control
        }
short TCmovs; //Cantidad de movimientos hasta el control

..............................

if (TCflag)
{ if ((TiempoControl.movimientos[bando] + 3) != 0)   TiempoRespuesta =
(TiempoControl.reloj[bando]) /
        (TiempoControl.movimientos[bando] + 3) -
        TiempoOperador; //***1***
 else
  TiempoRespuesta = 0;
 TiempoRespuesta += (TiempoRespuesta * TiempoControl.movimientos[bando])
 / (2 * TCmovs + 1); //***2***
}else TiempoRespuesta = NIVEL; //***3***
if ((iop == 2) || (investigacion))
 TiempoRespuesta = 99999; //No hay límite de tiempo
if (Sdepth > 0 && root->valor > Zvalor - zwndw) TiempoRespuesta -= ft; //***4***El
permanent brain ha estado funcionando
else
if (TiempoRespuesta < 1) TiempoRespuesta = 1;//Da como mínimo un segundo para
pensar
TiempoExtra = 0; //por defecto es cero

/* *1* podemos ver cómo se hace una división proporcional del tiempo y jugadas restan-
tes, descontando el tiempo del operador.
*2* TiempoRespuesta es tanto mayor cuanto mayor sea TiempoControl.movimientos,
(justo en la apertura o medio juego que son las fases iniciales de la partida).
*3* NIVEL da los segundos de respuesta, constituye un control constante.
*4*ft es la cantidad de tiempo que ha estado pensando la máquina durante el tiempo del
contrario,
la pregunta Sdepth > 0 indica que el análisis permanente ha llegado a alguna conclusión.
*/
```

estamos cerca del límite de tiempo y no ampliamos el plazo podemos perder si la máquina no llega a la jugada salvadora Te8 que mantiene la ventaja para las negras.

Así pues una jugada conflictiva es aquella en la que la variante principal prevista cae por debajo del alfa, para estos casos se asigna un plus de tiempo que puede llegar a ser el triple del inicialmente asignado.

Otra característica particular del ajedrez es que el grado de complejidad de una partida suele ser máximo en el medio juego (hablamos en términos generales) por ello se concede un tiempo respuesta extra para las jugadas correspondientes a esta fase del juego.

Jugadas obvias pueden ser consideradas aquellas que son fácilmente encontradas en las primeras iteracciones de búsqueda y que de no ser efectuadas conducen a situaciones perdidas. Entre estos casos podemos encontrar las recapturas que equilibran el material, aún así existen posiciones excepcionales y para no caer en errores el austríaco Christian Donninger aconseja efectuar un análisis mínimo hasta profundidad 5 antes de dar la respuesta. Esto lo dijo hace 3 años para procesadores 486 y programas rápidos basados en el turno nulo, hoy en día seguramente habría que añadir uno o dos pliegues más de profundidad.

El tiempo de respuesta se amplía en los momentos del juego de mayor complejidad

Veamos en un diagrama cómo ajustaría su tiempo un algoritmo de selección de jugadas basado en una búsqueda de árbol mediante sucesivos incrementos de profundidad (profundidad = Sdepth).

Este es un ejemplo en C, extraido de una versión particular del GNU, acerca de cómo se podría hacer una distribución del tiempo:

Aprovechar el tiempo de reflexión del contrario es algo que realizan todos los programas conocidos, el ordenador después de hacer su movimiento continúa el análisis suponiendo que el rival efecturá la jugada prevista como buena. A esta facultad se la suele denominar "permanent brain". Se han dado casos de "aprovechamiento" extremos en los que el ahorro de tiempo con respecto al rival ha llegado a ser un 70%. Sin embargo esta fuerte ventaja de tiempo (por ejemplo de 30 minutos) pocas veces es decisiva si la situación no es de ventaja, por lo general los tiempos de ambos jugadores tienden a equilibrarse en pocos movimientos (movimientos de "compensación"), aquellos en los que el bando con más tiempo no logra predecir las jugadas

de su rival y esto es algo que tarde o temprano ocurre, si durante esta fase la ventaja no se materializa ésta se pierde. Cuando se juega sin "permanent brain" entramos en el modo de juego "easy game", denominado así porque el ordenador pierde fuerza de juego.

Una pequeña triquiñuela legal usada por algunos jugadores para ahorrar tiempo (escasos segundos) se da cuando un programa ha acertado con el movimiento del rival mientras estaba en "permanent brain". Aquí basta con pulsar la tecla Intro (confirmamos que lo esperado se ha producido) para dar de alta la jugada en vez de introducirla completa que serían al menos 4 pulsaciones de teclado. Este detalle es de utilidad en las partidas rápidas donde el tiempo constituye en muchos casos el factor primordial para obtener la victoria.

Consumir menos tiempo que el rival es algo ciertamente complejo a nivel práctico, por lo general siempre existe un desfase que tiende a dar más tiempo al propio bando y menos al contrario. Es muy común el hecho de asignar un tiempo de operador a cada jugada realizada, este tiempo oscila entre los 3 y los 5 segundos. Algunos programas se encuentran provistos de mecanismos de rectificación del tiempo, como el de preguntar directamente durante la partida y en el periodo del rival mediante una ventana por el tiempo restante para ver si existen desfases. Durante el mundial de París (1997) el Kallisto holandés y el Junior israelí utilizaban "disimuladamente" las teclas + y – para añadir o quitar minutos a su tiempo restante (siempre durante el período de reflexión del contrario). Curiosamente nadie protestó por esta "invisible" novedad que para el próximo año hará con toda seguridad acto de presencia en cada uno de los programas participantes.

Destaca que en la fase de apertura, aquella en la que los dos programas juegan de un modo automático, los relojes no se usan hasta que uno de los dos se sale de su libro de teoría y comienza a pensar. Esto no se produce entre jugadores humanos.

Listado 2. Punto 2 de la figura 2.

```
  Comprobación tras cada iteracción.
 */

 if (valor > Zvalor - zwndw && valor > Tree[1].valor + 250)
   TiempoExtra = 0;
 else if (valor > Zvalor - 3 * zwndw)
   TiempoExtra = TiempoRespuesta;
 else
   TiempoExtra = 3 * TiempoRespuesta;

 /*valor contiene el resultado de la última iteración, si éste es excesivamente bajo
TiempoExtra se ve incrementado dependiendo del grado de "caida" de valor. zwndw equivale a
"zwndw = 20 + abs (Zvalor / 12)"
 */

 /****************************************/

 /* EXTRACTO DE void ElapsedTime (short iop).
   Comprobación cada x nodos (en este ejemplo x = 6000)
 */

 et = time ((long *) 0) - time0;
 if (et < 0)
   et = 0;
 ETnodes += 6000;
 if (et > et0 || iop == 1)
 {if (et > TiempoRespuesta + TiempoExtra && Sdepth > 1)
     flag.timeout = true;
   et0 = et;
   if (iop == 1)
   {time0 = time ((long *) 0);
     et0 = 0;
   }

   if (et > 0)
     evrate = NodeCnt / (et + ft);
   else
     evrate = 0;
   ETnodes = NodeCnt + 6000;
   UpdateClocks ();
 }

 /*flag.timeout indica si estamos fuera del tiempo permitido, obsérvese que nunca puede
tener el valor true antes de haber
   completado la primera iteracción ("&& Sdepth > 1").
   NodeCnt es el contador de nodos realizados, cada vez que NodeCnt > ETnodes se llama
a este extracto de código donde el valor de
   ETnodes es incrementado en 6000.
   iop tiene valor 0 cuando se le llama desde el analizador.
 */
```

Listado 3.

```
void evaluate(side,node,ply,depth,alpha,beta,nxtline)
short side,ply,alpha,beta;
unsigned short nxtline[];
struct leaf *node;

{
short xside,s,x,t;

  hung[white] = hung[black] = 0;
  xside = otherside[side];
  ataks(xside,atak[xside]); //**1** llamada a la rutina que coloca las fuerzas de las piezas
sobre los tableros de ataque

/*PieceList [side][0] contiene la casilla del rey del banbo side (posrey), si atak [xside] [pos-
rey] > 0 ello siginifica que el rey está en jaque y node adquiere el valor de mate "ply – 10000"*/

  if (atak[xside][PieceList[side][0]] > 0) {
     node->score = ply-10000;
     node->flags |= incheck;
     node->flags |= exact;
   }
  else
    {
     ataks(side,atak[side]);
     /*observese que primero se ha calculado ataks (xside..) en *1* por si el rey estuviera
en jaque lo cual hace innecesario calcular
        ataks  (side..) con el consiguiente ahorro de tiempo.*/

     if (atak[side][PieceList[xside][0]]) node->flags |= check;
     if (ply > Sdepth) t = 0; else t = 90;
     s = -PPscore[ply-1]+mtl[side]-mtl[xside];

     /*Obsérvese que ScorePosition (la auténtica función de evaluación) sólo es llamada si
se cumplen las siguientes condiciones:

     1º un margen razonable entre el valor de referencia alpha y el valor comparativo s+t basa-
do en el material existente sobre el tablero. Comparándolo con otras versiones cuando ply <=
Sdepth siempre se evaluaba. Sdepth es la profundidad nominal de análisis y depth la profundidad
restante.

     2º el jaque "node->flags & check".

     3º avance de un peón pasado "node->flags & pwnthrt".
     */

     if (s+t > alpha || (node->flags & check) ||
       (node->flags & pwnthrt)) ScorePosition(side,&s);
    PPscore[ply] = s-mtl[side]+mtl[xside]; //actualización de Ppscore para el siguiente turno
     if (s < alpha || depth > 1)
       {
        if (node->score < -12000) node->score = s;
       }
     else
       {
        ykillr = xkillr;
        CaptureSearch(xside,side,ply+1,9,-s-1,-alpha,s,&x,nxtline); //**2**
        node->score = -x;
        node->reply = nxtline[ply+1];
       }
     repetition(node);
    }
}
```

Kiss Chess

John Stanback es un reconocido programador de ajedrez autor del Zarkov y coautor del GNU. Recientemente se ha publicado en Gambitsoft su pequeño programa Kiss Chess escrito en C. El generador de movimientos es excesivamente lento como para ser tomado como modelo, pero el tratamiento que le confiere a las evaluaciones y su modo de ver las búsquedas de capturas resultan de interés.

Veamos su rutina de evaluación:

En *2* se ha producido una llamada a la búsqueda de capturas "CaptureSearch" desde la propia rutina de evaluación. Esta llamada esconde una idea frecuentemente comentada por los programadores de alto nivel, el producir nodos considerados fuera del árbol de decisiones y que sirven sólo para hacer más exacto un determinado aspecto de una evaluación considerada como "estática". Este tipo de subanálisis acotados suelen tener en cuenta la participación de unas pocas piezas en un área del tablero. Su mayor utilidad reside ante todo en los finales en los que por ejemplo sólo interesan los movimientos de 2 ó 3 piezas concretas de un total de 7 u 8. También ha sido empleado para observar la posibilidad de determinados desarrollos de aperturas que normalmente no hubieran podido ser evaluados debido a las explosiones de posibilidades.

Siempre existe un desfase que tiende a dar más tiempo al propio bando y menos al contrario

Es evidente que este tipo de microanálisis pueden llegar a producir decisiones erróneas en sus primeras fases de desarrollo pero su depuración puede conducir a producir resultados altamente positivos.

La rutina de búsqueda de capturas, considerada aquí como un proceso aparte y no incluido en la propia rutina search (también denominada PVS), posee algunas ideas originales:

Observese en *1* que el valor de v viene dado por el valor de la pieza a ser capturada "value[board[t]]" más el valor posicional de ésta dado en "svalor [t]". Este pequeño detalle puede ser de vital importancia en determinadas posiciones ante todo de finales. El caso más simple puede ser el de un peón pasado a punto de coronar. Su valor material es de 100 puntos pero su valor posicional puede ser de 250 puntos (este caso tan extremo es imposible, al menos para un programa, en las piezas como caballo, alfil, etc...), lo cual puede decidir entre continuar o no el análisis de una línea significativa.

Los programas existentes en la actualidad poseen un sistema de control de tiempo seleccionable por el usuario

Cabe destacar que las llamadas a las rutinas MakeMove y UnMakeMove que se encargaban de avanzar y retroceder jugadas (puntos *2* y *3*) están en el propio proceso de un modo resumido y por tanto produciendo un significativo ahorro de tiempo.

Como rasgo negativo cabe señalar que "permite" la captura simbólica del rey contrario devolviendo un valor de mate si se diera el caso, lo que provoca una pequeña pérdida de tiempo para algunas situaciones de jaques.

El KissChess es un pequeño aliento de ideas frescas que conviene explorar no sólo para incrementar la fuerza de un programa sino también para sentir el placer de experimentar con nuevos conceptos por desarrollar.

Listado 4.

```
void CaptureSearch(short side,short xside,short ply,short depth,short alpha,
        short beta,short qscore,short *best,unsigned short bstline[])

{
register short j,f,t;
short v,q,pnt,tempb,tempc,pbst,sv;
unsigned short nxtline[30];
struct leaf *node;

  *best = -qscore; bstline[ply] = 0;
  CaptureList(side,xside,ply); //llamada al generador de movimientos sólo para capturas
  pnt = TrPnt[ply]; pbst = 0;
  while (pnt < TrPnt[ply+1] && *best <= beta)
   {
    node = &Tree[pnt]; pnt++;
    f = node->f; t = node->t;
    v = value[board[t]]-qscore+svalue[t]; //**1**

    //si v no es mayor que la referencia de búsqueda alpha a pesar de la captura realizada
    entonces la variante no merece la pena.

    if (v > alpha)
    {
      if (board[t] == king) node->score = 10000;
      else if (depth == 1) node->score = v;
      else
       {
        ykillr = t; NodeCnt++; //**2**
        sv = svalue[t]; svalue[t] = svalue[f];
        tempb = board[t]; tempc = color[t];
        UpdatePieceList(tempc,t,1);
        board[t] = board[f]; color[t] = color[f];
        Index[t] = Index[f]; PieceList[side][Index[t]] = t;
        board[f] = no_piece; color[f] = neutral;
        CaptureSearch(xside,side,ply+1,depth-1,-beta,-alpha,v,
               &q,nxtline);
        node->score = -q;
        board[f] = board[t]; color[f] = color[t]; //**3**
        Index[f] = Index[t]; PieceList[side][Index[f]] = f;
        board[t] = tempb; color[t] = tempc;
        UpdatePieceList(xside,t,2);
        svalue[f] = svalue[t]; svalue[t] = sv;
       }
     if (node->score > *best)
            {
       pbst = pnt;
       *best = node->score;
       if (*best > alpha) alpha = *best;
       for (j = ply; j < 30; j++) bstline[j] = nxtline[j];
       bstline[ply] = (f<<8) + t;
      }
     }
   }
  if (pbst == 0) Ckillr[side] = -1;
   else Ckillr[side] = (Tree[pbst].f<<8) + Tree[pbst].t;
}
```

VISUAL C++

Mapas de mensajes (II)

Xesco Hernández (xesco@redestb.es)

Las MFC disponen de macros de mapas de mensajes que permiten dirigir distintos comandos hacia una misma función de respuesta, utilizar mensajes de usuario o actualizar la interfaz de la aplicación.

Después de haber visto cómo funcionan los mapas de mensajes y cómo se usan las macros más habituales, veremos algunas macros de uso menos común pero que pueden ser de gran utilidad.

ON_COMMAND_EX

Esta macro permite que una única función responda a diferentes comandos de menú recibiendo como parámetro el identificador del comando. No es posible escribir esta macro usando Class Wizard por lo que deberemos escribir el código necesario, de forma manual, fuera de los comentarios del Wizard. Resulta útil en los casos en que necesitamos que diferentes comandos de menú llamen a la misma función y necesitamos, además, distinguir dentro de la función qué comando en particular ha generado la llamada.

Podríamos pensar que con Class Wizzard y la macro ON_COMMAND también podemos hacer que diferentes IDs de menú llamen a la misma función. Veamos que esto es así, pero que el resultado nos lleva a un callejón sin salida. Efectivamente, podemos abrir ClassWizzard, selecionar la clase que queremos que responda a los comandos de menú, seleccionar el identificador del comando, seleccionar "COMMAND" y pulsar el boton Add Function. Class Wizard nos propondrá un nombre para la función a la que podemos llamar, por ejemplo, OnOpcion(). Para hacer que otro comando de menú llame a la misma función basta con repetir estos mismos pasos, seleccionando el nuevo identificador de menú, y cambiar el nombre de la función que propone el Wizard para darle el mismo nombre OnOpcion(). Ver figura 1.

Al seguir este procedimiento Class Wizard escribirá dos entradas ON_COMMAND en el mapa de mensajes, pero sólo declarará e implementará la función OnOpcion() una vez.

El resultado será:

```
BEGIN_MESSAGE_MAP(CMainFrame,
    CFrameWnd)
//{{AFX_MSG_MAP(CMainFrame)
ON_COMMAND(ID_OPCION1, OnOpcion)
ON_COMMAND(ID_OPCION2, OnOpcion)
ON_COMMAND(ID_OPCION3, OnOpcion)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CMainFrame::OnOpcion()
{
}
```

Las tres opciones de menú llamarán a la misma función OnOpcion() y ejecutarán el mismo código, es decir, las tres opciones harán lo mismo. Si este era nuestro objetivo hubiera sido más sencillo

dar el mismo identificador a las tres opciones y escribir la macro una sóla vez. Tal como lo hemos hecho no hay manera de distinguir dentro de la función OnOpcion() qué identificador en concreto ha pulsado el usuario.

Es en este tipo de situaciones donde debemos usar ON_COMMAND_EX.

El prototipo de esta macro y su función de respuesta debe ser:

```
ON_COMMAND_EX (ID, funcion)
afx_msg BOOL funcion(UINT nID);
```

Para dirigir los comandos ID_OPCION1, ID_OPCION2 e ID_OPCION3 hacia una misma función de respuesta OnOpcion() realizamos los siguientes pasos:

1. Declaramos la función como miembro de la clase fuera de los comentarios del Wizard.
2. Insertamos, en el mapa de mensajes de la clase, una macro ON_COMMAND_EX para cada opción de menú (también fuera de los comentarios del Wizard).
3. Implementamos la función de respuesta.

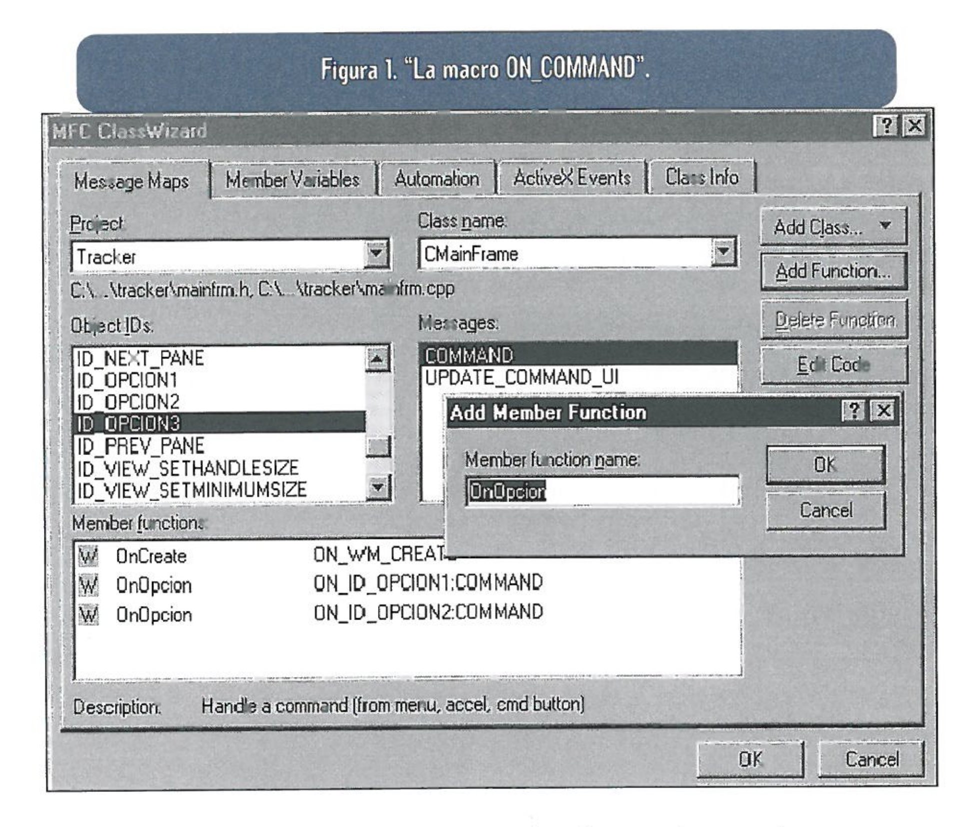
El listado 1 nos muestra el código necesario.

Ahora la función OnOpcion recibe como parámetro el identificador del menú que se haya pulsado y en función de este podemos realizar unas acciones u otras.

La función de respuesta debe devolver un valor booleano. Devolveremos TRUE si hemos procesado el mensaje o FALSE si no lo hemos procesado, lo que indica a las MFC que deben seguir la ruta de comandos para buscar funciones de respuesta en otras clases. Esto permite que si en algunos casos no queremos procesar el mensaje, devolvemos FALSE y permitimos que las MFC sigan buscando, según la ruta de comandos.

Como se puede observar debemos escribir tantas macros ON_COMMAND_EX como comandos de menú queramos dirigir a la función.

Figura 1. "La macro ON_COMMAND".



ON_COMMAND_RANGE

Esta macro permite que distintos comandos, con identificador dentro de un rango especificado, se dirijan a una única función de respuesta. Esta función recibirá como parámetro el identificador del comando que generó la llamada. Conseguimos el mismo efecto que usando ON_COMMAND_EX, con la ventaja de que sólo tenemos que escribir una entrada en el mapa de mensajes.

El prototipo de esta macro y de la función de respuesta es:

```
ON_COMMAND_RANGE(ID_CMD_1,
    ID_CMD_N, funcion)
afx_msg void funcion (UINT nID);
```

A modo de ejemplo, en el listado 2, hemos reescrito el mismo caso que en el ejemplo anterior usando esta nueva macro.

Para que esta macro funcione correctamente debemos asegurarnos que los valores de los identificadores ID_OPCION1 a ID_OPCION3 son números correlativos, ya que si hubiera algún otro identificador con valor dentro del rango señalado en la macro, también se dirigiría a OnOpcion().

Para conseguir que estos identificadores sean consecutivos podemos editar el archivo "resource.h", donde AppStudio define los valores de los identificadores, y editarlo para que sea así. Una solución mejor es definir estos identificadores fuera de "resource.h" en un fichero que sea de sólo lectura para AppStudio, con lo que podremos usar estos defines en los recursos, pero evitaremos que AppStudio les asigne valores desconocidos. Además, esto nos permitirá operar directamente con los valores de los identificadores.

Veamos un ejemplo. Tenemos una opción de menu "Zoom" con cuatro submenús: 25%, 50%, 75% y 100%. En lugar de dejar que AppStudio ponga los

Listado 1. "La macro ON_COMMAND_EX".

```
//{{AFX_MSG(CMainFrame)
...
//}}AFX_MSG
afx_msg BOOL OnOpcion(UINT nID);

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
//{{AFX_MSG_MAP(CMainFrame)
...
        //}}AFX_MSG_MAP
        ON_COMMAND_EX(ID_OPCION1, OnOpcion)
        ON_COMMAND_EX(ID_OPCION2, OnOpcion)
        ON_COMMAND_EX(ID_OPCION3, OnOpcion)
END_MESSAGE_MAP()

BOOL CMainFrame::OnOpcion(UINT nID)
{
        CString sTxt;
        switch(nID)
        {
        case ID_OPCION1:
                sTxt = "Respuesta a Opcion1";
                break;
        case ID_OPCION2:
                sTxt = "Respuesta a Opcion2";
                break;
        case ID_OPCION3:
                sTxt = "Respuesta a Opcion3";
                break;
        }
        AfxMessageBox(sTxt);
        return TRUE;
}
```

defines de estos identificadores en resource.h, creamos un archivo nuevo donde los definiremos y lo guardamos como "mires.h":

```
// mires.h
#define ID_ZOOM_25 35000
#define ID_ZOOM_50 35001
#define ID_ZOOM_75 35002
#define ID_ZOOM_100 35003
```

El valor de estos identificadores no debe interferir con valores que estén siendo utilizados por otros comandos. Podemos ver qué valores asignar consultando la nota técnica "TN020 ID Naming and Numbering Conventions", donde veremos que los identificadores de comandos deben estar en el rango 32768 a 57343. Dado que AppStudio empieza a enumerar los ID que creamos gráficamente a partir de 32768 dejamos un margen para que lo use AppStudio y empezamos a numerar en 35000.

Para que AppStudio pueda ver estos identificadores pero no modificarlos debemos ir al menú Ver, Resource Includes, y en la sección "Read-Only symbol directives" añadir nuestro archivo de cabecera "mires.h". Ver figura 2.

Hecho esto podemos usar el editor de recursos para añadir al menú de nuestra aplicación el menú zoom y sus cuatro submenús.

Para responder a las cuatro opciones del menú Zoom debemos escribir código en tres puntos (Ver listado 3).

1. Declarar una función miembro de la clase vista.
2. Insertar la macro ON_COMMAND_RANGE en el mapa de mensajes.
3. Implementar la funciónn de respuesta OnZoom

Como vemos en la función OnZoom, la macro ON_COMMAND_RANGE y el uso de identificadores consecutivos ayuda a escribir un código reducido y compacto.

Funciones de actualización de la interfaz de usuario

Estas funciones se utilizan para habilitar/inhabilitar y marcar/desmarcar los elementos de la interfaz de usuario como opciones de menú, botones de barras de herramientas y paneles de la barra de estado. Por ejemplo, para inhabilitar la opción de menú "Archivo-Guardar" cuando no tenemos ningún documento abierto.

Para asociar una función de actualización a un elemento de la interfaz de usuario disponemos de la macro ON_UPDATE_COMMAND_UI, cuyo prototipo es:

```
ON_UPDATE_COMMAND_UI( id, funcion)
afx_msg void funcion ( CCmdUI* pCmdUI );
```

Esta macro se puede insertar automáticamente con Class Wizard, seleccionando el identificador de la opción de menú y "UPDATE_COMMAND_UI". Con esto Class Wizard habrá escrito, en

Listado 2. "La macro ON_COMMAND_RANGE".

```
//{{AFX_MSG(CMainFrame)
...
//}}AFX_MSG
afx_msg void OnOpcion( UINT nID );

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ...
        //}}AFX_MSG_MAP
        ON_COMMAND_RANGE(ID_OPCION1, ID_OPCION3, OnOpcion)
END_MESSAGE_MAP( )

void CMainFrame::OnOpcion(UINT nID)
{
        // igual que antes
}
```

nuestra clase, una función desde la que podremos actualizar este elemento de menú. Esta misma función será llamada para actualizar todos los elementos que tengan el mismo identificador. Así, si existe una opción de menú con identificador ID_ABRIR y un botón de la barra de herramientas con el mismo identificador ambos se actualizarán en la función OnUpdateAbrir().

Estas funciones de actualización son llamadas por el marco de la aplicación y reciben como parámetro un puntero a un objeto de la clase CCmdUI. Esta clase tiene los métodos Enable(), SetCheck(), SetRadio() y SetText()que nos permitirán interactuar con el elemento de la interfaz de usuario al que hace referencia la macro. Así, para inhabilitar la opción de menú ID_ABRIR, cuando no hay ningún documento abierto, deberemos asociarle una función de actualización y usar el puntero pCmdUI para llamar a Enable():

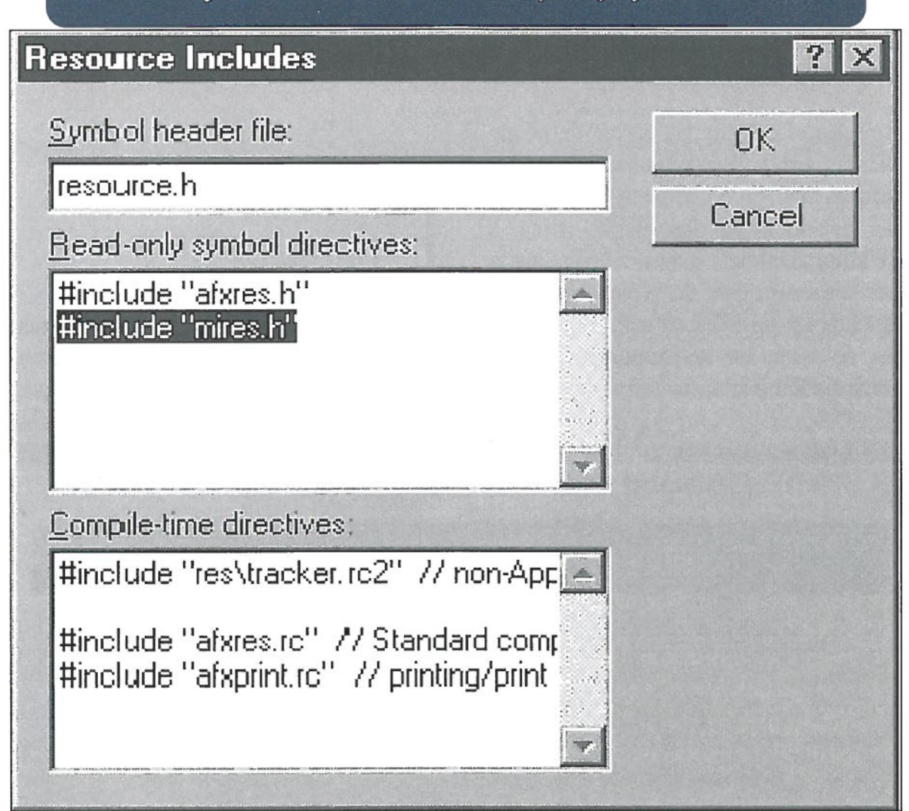
```
void CMyView::OnUpdateAbrir(CCmdUI*
        pCmdUI)
{
        pCmdUI->Enable(bHayDoc);
}
```

Donde bHayDoc será una variable booleana que tendrá el valor TRUE o FALSE según queramos habilitar/inhabilitar la opción de menú.

La función Enable tendrá distinto efecto según el tipo de elemento que se esté actualizando. En la tabla 1 podemos ver el efecto de las funciones de la clase CCmdUI sobre distintos elementos de la interfaz de usuario.

Como hemos dicho estas funciones son llamadas por el marco de la aplicación. En el caso de una opción de menú, la función de actualización es llamada cuando el usuario selecciona el ítem del menú principal antes de que sus opciones aparezcan desplegadas en pantalla. Para conseguir esto las MFC interceptan el mensaje WM_INITMENUPOPUP que se genera cuando el usuario va a desplegar un menú. En respuesta a este mensaje, las MFC envían un comando de actualización para cada una de las opciones del menú desplegable. Una vez enviado el comando, cualquier clase que esté en la ruta de comandos podrá interceptarlo si tiene en su mapa de mensajes la macro ON_UPDATE_COMMAND_UI adecuada. Esto llamará a la función de actualización que habilitará/inhabilitará la opción de menú. A continuación se mostrará el menú desplegable con sus opciones debidamente actualizadas.

Figura 2. "Identificadores mantenidos por el programador".



Listado 3. "Identificadores consecutivos".

```
//{{AFX_MSG(CMyView)
...
//}}AFX_MSG
afx_msg void OnZoom(UINT nID);

BEGIN_MESSAGE_MAP(CMyView, CView)
        //{{AFX_MSG_MAP(CMyView)
        ...
        //}}AFX_MSG_MAP
        ON_COMMAND_RANGE(ID_ZOOM_25, ID_ZOOOM_100, OnZoom)
END_MESSAGE_MAP( )

void CMyView::OnZoom(UINT nID)
{
        int nOpcion = nID-ID_ZOOM_25; // 0, 1, 2, 3
        int nZoom = 25*nOpcion + 25;  // 25, 50, 75, 100
        PonZoom(nZoom);
}
```

Si el marco de la aplicación, a lo largo de la ruta de comandos, no encuentra ninguna macro ON_UPDATE_COMMAND_UI asociada con la opción de menú este se habilitará si existe una macro ON_COMMAND correspondiente. Si no existe ON_COMMAND la opción de menú se inhabilitará.

En el caso de las barras de herramientas estas son actualizadas en los tiempos muertos de la aplicación. El bucle de mensajes de las MFC llama a CWinApp::OnIdle() cuando no hay ningún mensaje en la cola de la aplicación. En esta función las MFC actualizan los botones de barra de herramientas de una forma similar a la usada para los menus.

Existe, también, la macro ON_UPDATE_COMMAND_UI_RANGE que permite que un conjunto de elementos tengan la misma función de actualización. Al igual que sucedia con ON_COMMAND_RANGE estos elementos deben tener sus identificadores dentro de un rango determinado y sus valores deben ser consecutivos.

Mensajes de usuario

Los mensajes de usuario se comportan como los mensajes Windows con la diferencia de que son mensajes definidos por nosotros. Son útiles cuando necesitamos que una ventana envíe a otra un mensaje propio, no predefinido en Windows.

Para responder a estos mensajes disponemos de la macro:

```
ON_MESSAGE( <message>, <memberFxn> )
afx_msg LONG memberFxn( UINT, LONG );
```

Antes de ver cómo utilizar los mensajes de usuario repasemos cómo enviar mensajes a una ventana. En todas las MFC disponemos de dos funciones para enviar mensajes, SendMessage() y PostMessage(). Estas funciones son miembros de CWnd y envían un mensaje a la propia ventana. SendMessage envía el mensaje directamente a la ventana, llamando a su Window Procedure. El mensaje se atiende inmediatamente y la función no retorna hasta que ha finalizado el proceso de respuesta al mensaje. PostMessage pone el mensaje en la cola de mensajes de la aplicación y retorna inmediatamente. Cuando el bucle de mensajes llega a leerlo lo pasa a la ventana correspondiente y es procesado.

Disponemos de dos funciones para enviar mensajes, SendMessage() y PostMessage()

El prototipo de SendMessage(), igual que el de PostMessage() es:

```
LRESULT SendMessage(UINT message, WPARAM wParam=0, LPARAM lParam=0);
```

Sus parámetros son:

- **message:** identificador del mensaje. Puede ser un mensaje predefinido de Windows como WM_XXX o WM_COMMAND o un mensaje de usuario.
- **wParam:** contiene información adicional, su significado depende del mensaje.
- **lParam:** contiene información adicional, su significado depende del mensaje.

Por ejemplo, si quisieramos cerrar

Tabla 1. "Actualizar la interfaz de usuario".

	Opción de menú	Botón barra herramientas	Panel barra de estado
Enable	habilita / inhabilita	habilita / inhabilita	muestra / oculta texto del panel
SetCheck	marca / desmarca	mantiene pulsado	pone borde pop-up o normal
SetRadio	marca usando (.)	igual que SetCheck	igual que SetCheck
SetText	pone texto del menú	no aplicable	pone texto del panel

una ventana a la que apunta pWnd podríamos enviarle un mensaje WM_CLOSE haciendo:

```
pWnd->SendMessage(WM_CLOSE);
```

El mensaje WM_CLOSE no usa para nada los parámetros wParam, lParam, que por defecto toman el valor cero.

Si quisieramos provocar la misma acción que tendría lugar cuando un usuario eligiera una opción de menú ID_OPCION1, hariamos:

```
pWnd->SendMessage(WM_COMMAND,
    ID_OPCION1);
```

En este caso el mensaje WM_COMMAND utiliza el parámetro wParam para indicar el identificador del comando que se está enviando.

Si estamos en un diálogo y un control de edición IDC_EDIT1 cambia de contenido, el diálogo recibiria un mensaje WM_COMMAND con IDC_EDIT1 en LOWORD de wParam, EN_CHANGE en HIWORD de wParam y el Handle del control en lParam.

Hay que remarcar que WM_COMMAND es un mensaje predefinido en Windows que se usa tanto para mensajes de comandos, como para notificaciones de controles. No hay que confundirlo con la macro ON_COMMAND que usan las MFC para los comandos. La distinción entre mensajes Windows, mensajes de comandos y notificaciones de controles es algo que crean los mapas de mensajes de las MFC. El API de Windows, que define los mensajes existentes, no hace esta distinción.

Veamos un ejemplo de mensaje de usuario. Supongamos que en una aplicación necesitamos, repetidas veces, añadir un número al título de su ventana principal. Una posible solución sería definir un mensaje de usuario WM_PONNUMERO y enviarlo a la ventana principal pasándole el número que queremos añadir. La ventana principal, a través de su mapa de mensajes, responderá a este mensaje ejecutando la función de respuesta adecuada que añadirá el número al título de la ventana.

Listado 4. "Mensaje de usuario"

```
#define WM_PONNUMERO  (WM_USER+1)

afx_msg LONG OnPonNumero(WORD wParam, LONG lParam);

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ...
    //}}AFX_MSG_MAP
    ON_MESSAGE(WM_PONNUMERO, OnPonNumero)
END_MESSAGE_MAP( )

LONG CMainFrame::OnPonNumero(WORD wParam, LONG lParam)
{
        CString sNum, sTit;
        sNum.Format("%d", wParam);
        GetWindowText(sTit);
        SetWindowText(sTit + sNum);
        return 0;
}
```

Para que la clase ventana marco responda a este mensaje debemos realizar los siguientes pasos (Ver listado 4):

1. Definir el mensaje de usuario WM_PONNUMERO.
2. Declarar una función de respuesta miembro de la clase Ventana marco.
3. Escribir una entrada ON_MESSAGE en su mapa de mensajes.
4. Implementar la función de respuesta.

Para evitar asignar al mensaje de usuario un número que ya esté en uso por otros mensajes, las MFC reservan un rango de valores para los mensajes de usuario. El mensaje de usuario debe estar entre WM_USER y 0x7FFF (32767). En este ejemplo hemos utilizado WM_USER+1.

Para enviar este mensaje desde cualquier punto que queramos del código deberemos obtener un puntero a la ventana marco. Para esto utilizaremos GetParentFrame() si estamos en una vista, y llamaremos a SendMessage o PostMessage:

```
pFrame->PostMessage(WM_PONNUMERO, 5);
```

Cuando la ventana marco reciba este mensaje, llamará a la función OnPonNumero pasandole el valor 5 en el parámetro wParam. Esta función se encargará de añadir el número 5 al título de la ventana.

Hemos visto las bases del funcionamiento de los mapas de mensajes de Visual C++, cómo utilizar las macros más habituales y algunas técnicas más avanzadas para tratar con mensajes. Todo esto nos ayudará a comprender el funcionamiento interno de las MFC y a conocer los distintos métodos de que disponemos para trabajar con mensajes y funciones de respuesta.

El autor

Xesco Hernández es programador e imparte cursos en BIT formación informática. Para cualquier comentario referente a este artículo, el lector puede contactar con él en la dirección de correo. electrónico indicado.

SERVIDORES

Conectividad Windows-Unix con Samba (y II)

Juan José Taboada León, José Juan Mora Pérez
(abeniz@uhu.es)

En el pasado nº 41 realizamos un estudio de los pasos a seguir para la instalación y configuración del servidor Samba. Intentaremos en este artículo profundizar un poco más realizando una breve descripción de alguno de los parámetros de configuración, la herramienta Smbfs y la forma de utilizar nuestro servidor como servidor de impresión.

Samba como servidor de impresión

Uno de los asuntos que se quedaron pendiente en el artículo anterior fue la utilización del servidor Samba como servidor de Impresión.

Con la instalación de Samba conseguimos convertir nuestro sistema UNIX tanto en un servidor de ficheros como en un servidor de Impresión. Ya se comentó, que para acceder a cualquier servicio, éstos han de recogerse en el fichero smb.conf;. Algo que debe quedar totalmente claro es que Samba se limita a ser un intermediario entre el cliente SMB y el sistema de impresión de la máquina UNIX.

Para utilizar los servicios de impresión del Samba, todas las impresoras conectadas a la máquina UNIX que vayamos a utilizar, deberán estar perfectamente configuradas y funcionando correctamente. Si no estamos seguros del correcto funcionamiento de las impresoras, podríamos intentar realizar alguna prueba, imprimiendo un fichero por ejemplo:

```
lpr texto.txt.
```

Un fichero bastante importante a la hora de configurar los servicios de impresión del servidor Samba es el fichero **/etc/printcap**. En él, podremos encontrar una descripción detallada de todas las impresoras disponibles en nuestro sistema UNIX. Y es de donde Samba obtendrá la información necesaria para el proceso de impresión. En algunos sistemas tales como Solaris, este fichero no existe, por lo tanto tendremos que crearlo con los nombres de las impresoras presentes en el sistema. El nombre que tiene una determinada impresora es bastante importante, ya que será este nombre el indentificador que utilizaremos en los distintos servicios para referirnos a una u otra impresora.

La configuración de un servicio de impresión la podemos realizar utilizando la sección [printers].

De esta forma, se pueden utilizar los nombres de impresoras en las llamadas a los servicios, por ejemplo: \\localhost\nombre_impresora.

Samba buscará el nombre de impresora en **/etc/printcap**, si lo encuentra se creará de forma dinámica un nuevo servicio y copiará la sección [printers] dentro de este nuevo servicio. Si el nombre de impresora que utilizamos no corresponde a ninguna de las impresoras presentes en **/etc/printcap**, será rechazada la petición de conexión al servicio.

En la tabla 1, se muestra un ejemplo de servicio [printers]. Utilizamos las variables de sustitución %S, para que el

comando lpr mande los trabajos de impresión a la impresora cuyo nombre hemos utilizado como nombre de servicio. En nuestro ejemplo sería nombre_impresora.

Para comprobar cuáles son los nombres de impresoras válidos podemos utilizar el programa **testprns**, basta con pasarle como parámetro el nombre de la impresora y éste buscará en el fichero **/etc/printcap** si aparece alguna impresora con este nombre.

Otro modo de configurar un servicio de impresión es crear un servicio, en él situaremos los parámetros necesarios que mejor se ajusten a nuestras necesidades. En la tabla 2 aparece un ejemplo de un servicio de impresión.

Antes de desglosar el servicio de impresión de la tabla 2, debemos definir el valor del parámetro global **printing,** el cual determina el estilo de impresión. Este estilo dependerá del sistema UNIX que utilicemos. Samba soporta varios estilos de impresión, tales como: bsd, sysv, hpux, aix, plp, qnx. Si nuestro sistema utiliza **lpr** debemos utilizar el estilo bsd, si utiliza **lp** entonces el estilo será sysv.

La sección de la tabla 2 está compuesta por los siguientes parámetros:

El parámetro **path** cuyo valor es el directorio temporal, /tmp. Hay que tener cuidado en el momento de elegir el directorio del servicio, ya que éste debe ser un directorio en el cual todos los usuarios del servicio tengan privilegios de lectura/escritura.

El parámetro **printable** es el que da el carácter de servicio de impresión a un determinado servicio. Todos los servicios que sean de impresión deben utilizar este parámetro.

La utilización de los diferentes parámetros **print command, lpq command** y el parámetro **lprm command,** se encuentran descritos en la lista de parámetros de servicio que se enumeran más adelante.

En el caso de que no dispongamos de impresora, podemos realizar todo el proceso de configuración y enviar los trabajos a un directorio determinado para comprobar que el servicio funciona correctamente. Para ello podemos utilizar en la configuración del servicio el parámetro **print command** que permite cambiar el destino de los trabajos a imprimir. Un ejemplo podría ser el siguiente:

```
print command = cat %s > /tmp/print.%u.$$
```

Esta línea copia los ficheros que enviamos al servicio de impresión de Samba, al directorio /tmp y le cambia el nombre por el de print.%u.$$, donde **%u** se sustituye por el nombre del usuario que accede al servicio, y **$$** por el ID del proceso.

Usando una impresora compartida

Podemos utilizar el programa **smbclient** para acceder a cualquier impresora compartida en nuestra red. Para conseguir esto, necesitamos crear un pequeño *script* que realizará lo siguiente:

1. Copiar el fichero a imprimir a un directorio temporal.
2- Crear una cadena de caracteres formada por comandos del programa smbclient.
3- Llamar al programa sbmclient, pasándole la cadena construida anteriormente, así como el servidor y el servicio SMB que deseamos utilizar.

Todo este proceso lo realiza el script de la tabla 3. Una vez creado el *script* solo tenemos que ejecutar la siguiente orden:

```
cat fichero.txt | smbprint1 '\\localhost\impresora1'
```

Con este *script* únicamente pretendemos dar un ejemplo de su funcionalidad, es lo suficientemente básico para que todos aquellos lectores que no estén muy familiarizados con la programación de Shell, lo entiendan fácilmente.

Tabla 1: Ejemplo de sección [printers].

```
[printers]
path = /tmp
printable = yes
print command = lpr -r -P%S %p %s
```

Tabla2 : Ejemplo de un servicio de impresión.

```
[Impresora1]
comment = Impresora Local1
path=/tmp
printable = yes
print command = lpr -r -P %p %s
lpq command = /usr/bin/lpq %p
lprm command = /usr/bin/lprm -P %p %j
```

Smbfs

Smbfs es un paquete que no viene con la distribución de Samba, pero lo podemos conseguir en la Web de Samba. Hasta ahora hemos visto cómo acceder al sistema de ficheros de la máquina UNIX desde Windows o cómo acceder a un determinado recurso compartido en una máquina Windows desde UNIX. También hemos visto cómo utilizar las impresoras compartidas. Una de las causas que argumentamos para la instalación de Samba, fue la de conseguir una mayor operatividad a la hora de trabajar con máquinas UNIX y Windows, evitando el uso de programas tales como FTP.

Este objetivo sólo se ha conseguido en parte. Samba permite trabajar con directorios del sistema de archivos de la máquina UNIX, desde una máquina Windows, de forma fácil y cómoda, pero ¿qué ocurre con los usuarios de la máqui-

Tabla3 : Ejemplo de un script para imprimir.

```
#!/bin/sh
# Nombre : smbprint1
# Script para imprimir un fichero utilizando
# un servicio del servidor Samba.
# uso :                    cat ejemplo.txt | smbprint1 '\\servidor\servicio'

printfile = "/tmp/smbprint.$$"
servicio=$1

echo "Imprimiendo..."
cat > $printfile
if [ -s $printfile] ; then
(echo "translate" ; echo "print $printfile"; echo "quit") \
 | smbclient  $servicio -P -N
fi
exit 0
```

na UNIX?, ellos aún deben de enfrentarse con un programa como Smbclient.

Smbfs viene a solucionar este problema ya que con la utilización de Smbfs podemos montar un directorio compartido, perteneciente a un servidor SMB (Windows, Os/2, etc.) en el árbol de directorios del sistema de archivos UNIX. De esta forma no tenemos que utilizar Smbclient para trabajar con los recursos que están en otras máquinas. Dando mayor versatilidad y funcionalidad a la utilización de Samba.

Algo que no se nos puede pasar por alto es que si nos decidimos a instalar Smbfs, nos tenemos que asegurar que nuestro Kernel soporte el sistema de ficheros smbfs, (en el momento en el que estabamos escribiendo este artículo, smbfs sólo estaba disponible para LINUX). Entonces sólo tendremos que volver a compilar nuestro Kernel para que pueda soportar dicho sistema de ficheros o bien utilizar el sistema de módulos que LINUX nos ofrece.

Smbfs consta de 2 programas, **smbmount** y **smbunmount**, los cuales tienen la misma función que sus homónimos de UNIX (mount y unmount) con la particularidad de que éstos nos permiten montar recursos compartidos mediante SMB. Su sintaxis es la siguiente:

```
smbmount //PC-WIN/DIR1 /usr/local/samba/dir1 [-
    P password]
```

Smbmount necesita que le pasemos el recurso al que queremos acceder, el directorio en el cual vamos a montar el recurso y el password que no es necesario. Si no se lo facilitamos, nos lo pedirá más tarde. Smbmount cuenta con algunos parámetros que una vez instalado podremos consultar en el manual.

```
smbunmount /usr/local/samba/dir1
```

Smbunmount consta de un único parámetro, el directorio que vamos a desmontar.

Parámetros

Vamos a realizar una breve descripción de algunos parámetros de configuración del servidor Samba.

Entonces dividiremos los parámetros en globales y de servicio. Recordemos que todos los parámetros dʃ e servicios pueden ser utilizados en la sección global, pero los parámetros globales no pueden ser utilizados en las secciones de servicio.

- **Globales**

- **dead Time** = número
  Este parámetro nos permite configurar el tiempo de inactividad que el servidor Samba, espera antes de cortar la conexión. El valor del parámetro está en minutos. La utilización de este parámetro, nos permite evitar que se agoten los recursos del servidor Samba cuando se han establecido un gran número de conexión inactivas.

- **default service** = nombre_servicio
  Si no se ha podido establecer una conexión con un determinado servicio, con la utilización de este parámetro, conseguimos que el cliente acceda al servicio especificado en el parámetro, siendo éste el servicio por defecto.

- **log file** = fichero
  Con este parámetro podemos especificar el nombre del fichero que se utilizará como fichero de registro en una determinada conexión. Esto nos permitirá crear distintos ficheros de registro utilizando variables de sustitución, pudiendo tener un fichero de registro por cada usuario que acceda al servidor Samba de la siguiente forma.

```
log file = /usr/local/samba/lib/log_%u
```

- **log level** = número
  El valor de este parámetro permite especificar el nivel de registro que deseamos tengan los ficheros de registro.

- **printing** = estilo
  Controla la forma en la cual la información de estado de la impresora es tratada por el sistema y afecta a otros parámetros como **print command, lpq command, lprm command**. Existen 6 estilo, bsd, sysv,

hpux, aix, plp y qnx. Estos estilos dependen del sistema de impresión de la máquina UNIX, si utilizamos lpr el estilo debe ser bsd, si por el contrario tenemos lp debemos utilizar el estilo sysv.

- **root directory** = /dir
  Se especifica el directorio que Samba tomará como directorio raíz De esta forma se intentará evitar- que se puedan acceder a directorios anteriores.

root directory = /usr/local/samba

- **wins support** = yes/no
  Si queremos que nuestro servidor Samba actúe como un servidor WINS, podemos habilitar esta opción mediante este parámetro.

- **wins server** = nombre DNS
  Con este parámetro podemos especificar el nombre de otra máquina, la cual actúa como servidor WINS.

- **workgroup** = Nombre
  El valor que toma este parámetro es el nombre que recibe el servidor Samba, con el cual se identificará en la red.

- Servicios

- **allow hosts** = lista hosts
  Con este parámetro podemos crear una lista de hosts a los cuales se les permite el acceso a un determinado servicio. De esta forma conseguimos limitar el acceso al servicio ya que únicamente éstos tendrán acceso. Podemos emplear tanto el nombre como el IP. Tenemos la opción de referirnos a un grupo de IP, utilizando únicamente sus números más significativos.

allow hosts = 158.167. Se permite el acceso a todos los hosts cuyos IP sean 158.167.*.*

También podemos hacer uso de EXCEPT para limitar aún más la lista.

allow hosts = 158.167. EXCEPT 158.167.124.100 Se permite el acceso a todos los hosts cuyos IP sean 158.167.*.* excepto al IP 158.167.124.100.

- **available** = yes/no
  Activa/desactiva es un servicio determinado, de esta forma dicho servicio no podrá aceptar peticiones de conexión.

- **browseable** = yes/no
  Permite que seleccionemos entre mostrar o no, un servicio determinado en la lista de servicios disponibles. Este parámetro es bastante útil cuando queremos que la existencia de un determinado servicio únicamente sea conocida por un grupo de usuarios.

- **copy** = nombre_servicio
  Con este parámetro podemos copiar un servicio, dentro de otro. De esta forma podemos crear plantillas de servicios y utilizarlas según nuestras necesidades.

- **deny hosts** = lista hosts
  Este parámetro es exactamente igual que **allow hosts**, pero la finalidad es la contraria, ya que los IP que aparezcan en la lista tienen terminantemente prohibido el acceso al servicio.

- **dont descend** = /dir1,/dir2, ...
  El valor del parámetro es una lista de directorios los cuales el servidor Samba mostrará como directorios vacíos.

- **force group** = grupo
  Este parámetro toma como valor el nombre de un grupo, este grupo debe ser uno de los grupos del servidor UNIX. La finalidad de este parámetro es forzar a que todas las conexiones que se intenten realizar con el servicio, las realice alguno de los usuarios pertenecientes a dicho grupo. Si algún usuario de cualquier otro grupo intenta acceder al servicio, será rechazada su petición.

- **force user** = usr
  Con este parámetro no forzamos a que sea un usuario, de un grupo específico, el que tiene que acceder a un servicio determinado, sino que es un usuario determinado el único que tiene acceso a este servicio.

- **guest account** = usr
  Este parámetro es necesario cuando queremos crear un servicio para invitados o público. Cuando los invitados accedan a un servicio, a estos invitados se les debe conceder ciertos privilegios dentro del sistemas de archivos de la máquina UNIX. Con éste parámetros decidimos cuáles van a ser estos privilegios, ya que todos los invitados tendrán los mismos privilegios que tenga el usuario '**usr**' dentro del sistema UNIX. Una buena idea es crear un usuario con ciertos privilegios y evitar que se puedan entrar en el sistema UNIX con dicho usuario.

- **guest only**= yes/no
  Este parámetro limita el acceso a un determinado servicio, obligando a que todos los usuarios que intenten acceder a éste, sean usuarios invitados. Para que este parámetro tenga efecto, el servicio debe ser público.

- **invalid users** = usr1 usr2 ...
  Este parámetro también nos permite limitar el acceso a un determinado servicio, ya que los usuarios que pertenezcan a la lista de usuarios, no tendrán derecho a acceder al servicio.

- **lpq command** = comando
  Con este parámetro especificamos el comando que deseamos que se ejecute cuando realicemos una petición para la obtención de la información sobre la cola de impresión en la máquina UNIX. Este parámetro acepta dos variables de sustitución. La primera es %p, la cual será sustituida por el nombre de la impresora. La segunda es %j, que será sustitui-

da por en número del trabajo en la cola de impresión.

- **lprm command** = comando
Este otro parámetro se utiliza para especificar el comando que deseamos ejecutar cuando se recibe una petición para borrar un determinado trabajo de la cola de impresión. Permite utilizar las variable %p y %j, exactamente igual que con el parámetro lpq command.

- **max connections** = número
El valor de este parámetro es el número de conexiones simultáneas que se permiten en un servicio determinado. Si número es 0, se considera que el número de conexiones permitidas son infinitas. Podemos utilizar éste parámetros en aquellos casos en los que un número excesivo de conexiones perjudiquen las prestaciones que un servicio debe ofrecer.

- **path** = /dir
Especifica el directorio sobre el cual se trabajará cuando un usuario cualquiera acceda a un servicio determinado. Debemos asegurarnos que los privilegios en el sistema de archivos que tiene un directorio determinado, sean iguales a los que tienen los usuarios que accederán al servicio.

- **preeexec** = comando
Especifica el comando que se ejecutará justo después de comenzar una conexión al servicio. Las posibilidades de este parámetro son muchas. Un ejemplo podría ser el crear un servicio el cual, cuando un usuario accediera a él mediante este parámetro, copiáramos unos determinados ficheros dentro del directorio de este parámetro, a los cuales el usuario tendría la posibilidad de acceder. Si accediera cualquier otro usuario distinto copiáramos otros ficheros. Este ejemplo puede resultar algo tonto pero sólo es una pequeña muestra de la utilidad de este parámetro.

- **postexec** = comando
Especifica el comando que se ejecutará justo antes de terminar una conexión al servicio. Siguiendo con el ejemplo anterior, este comando podría ser el encargado de borrar los ficheros del directorio.

- **print command** = comando
Este parámetro es fundamental en un servicio de impresión, ya que con él especificamos el comando que se ejecutará cuando deseamos imprimir algo, utilizando un servicio determinado. Ya hemos dicho antes que Samba no es un administrador de impresión, sino que actúa de puente entre el cliente y el sistema de impresión presente en la máquina UNIX. Por lo tanto este parámetro tiene distintos valores según el sistema que utilicemos. El parámetro acepta dos variables de sustitución, %p es sustituido por el nombre de la impresora y %s que es sustituido por el nombre del fichero que deseamos imprimir.

- **public** = yes/no
Este es otro de los parámetros necesarios para la configuración de los servicios para invitados, ya que es el que le da el carácter de servicio público a un determinado servicio.

- **read list** = usuario1, usuario2,...
Los usuarios que aparezcan en la lista únicamente tienen acceso de lectura la servicio.

- **read only** = yes/no
Con 'read only', el servicio es únicamente para lectura, para todos los usuarios que intenten acceder.

- **valid users** = usuario1,usuario2 ...
Es el inverso del parámetro **invalid users**, ya que únicamente los usuarios que aparezcan en la lista son los usuarios válidos para acceder al servicio.

- **wide links** = yes/no
Controla el acceso a directorios con *links* que estén fuera del árbol de directorio pertenecientes al directorio que hemos establecido como directorio raíz con el parámetro **root directory**.

- **writable** = yes/no
Permite que los usuarios que tengan acceso al servicio y tengan permiso de escritura en el directorio del servicio, puedan escribir en dicho directorio.

- **write list** = usuario1,usuario2 ...
Los usuarios que estén presentes en la lista y tengan permiso de escritura en el directorio del servicio, pueden escribir en dicho directorio.

Conclusión

En este artículo terminamos de describir las principales características de Samba como servidor de ficheros y como servidor de impresión que iniciamos en el artículo anterior del pasado nº 41.

Se ha tratado la configuración de los servicios de impresión, el paquete Smbfs para LINUX y se han descrito algunos de los parámetros de configuración del servidor Samba.

Damos por concluida esta serie de dos artículos en la que hemos intentado poner en conocimiento del mayor número de lectores esta herramienta, que sin duda, estamos seguros que será una de las herramientas para compartir recursos a tener en cuenta en un futuro no demasiado lejano.

Si el lector desea ponerse en contacto con los autores, lo puede hacer a través de la dirección de correo abeniz@uhu.es. Nos gustaría conocer la opinión de todos aquellos lectores interesados en este tema, ya que estamos estudiando la posibilidad de en un futuro crear una lista de correo sobre Samba.

Computación Evolutiva

Manuel de la Herrán Gascón (mherran@usa.net)

La *Inteligencia Artificial* (IA) no sólo consiste en idear algoritmos y estructuras de datos para solucionar problemas. También trata acerca de la inteligencia humana, y por extensión, sobre la vida. Dentro de la IA, la *Vida Artificial* ofrece algunos mecanismos de resolución de problemas muy eficientes y originales. Además, toma muy en serio sus aspectos más filosóficos. ¿Qué tienen estas simulaciones de fantástico? Podemos sentirnos una especie de dios, observando y modificando a nuestro antojo un mundo poblado por seres virtuales [A1]. También es interesante jugar con la idea de que nosotros mismos somos las "hormigas", viviendo bajo los designios de *El Programador* [E1] [E2] [E3]. Pero no se trata sólo de eso. Se trata de que la Vida Artificial ofrece una nueva perspectiva sobre los problemas que afectan a cualquier grupo, como por ejemplo, la humanidad.

Este artículo completa la serie de los números 36, 37 y 38 y contiene la bibliografía correspondiente a los temas que se han abordado. Voy a profundizar en algunos aspectos prácticos de la *Computación Evolutiva* como método de resolución de problemas, pero empezaré por introducir la evolución en su forma más general. Si aún no has podido ejecutar alguna de estas simulaciones y tienes un PC, echa un vistazo al CD-ROM que viene con la revista y encontrarás el código fuente completo en Visual Basic 5.0 y el programa de instalación para Windows de "Ejemplos de Vida", con varios de estos mundos.

Vida y Evolución

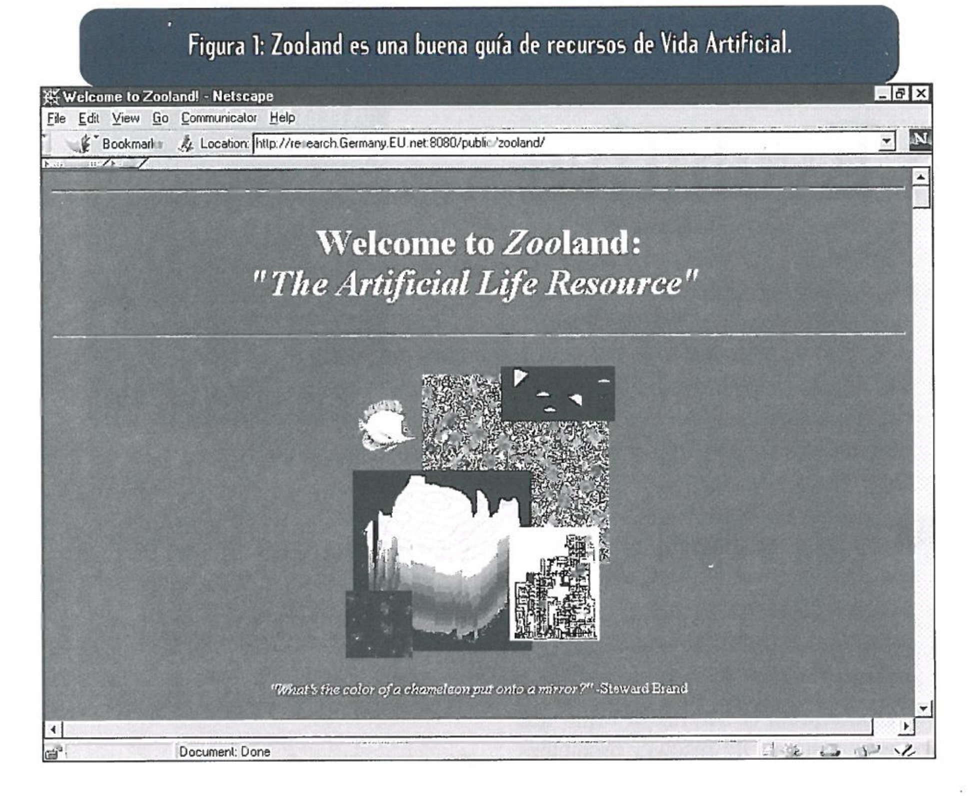
La *Vida Artificial* consiste en observar la vida natural e imitarla en un ordenador [A2] [A6]. La *Computación Evolutiva* [B11] interpreta la naturaleza como una inmensa máquina de resolver problemas y trata de encontrar el origen de dicha potencialidad para utilizarla en nuestros programas [B4] [B6]. Efectivamente, en la naturaleza todos los seres vivos se enfrentan a problemas que deben resolver con éxito, como conseguir más luz del sol, o cazar una mosca. Un programador con espíritu práctico no envidia la capacidad de la naturaleza para resolver problemas: la imita [B1].

En la naturaleza todos los seres vivos se enfrentan a problemas que deben resolver con éxito

El origen de esta capacidad está en la *evolución*, producida por la *selección natural*, que favorece la perpetuación de los individuos más adaptados a su entorno. Esto es lo que tenemos que simular. La principal diferencia conceptual entre la *selección natural* (o que se produce sin intervención del hombre) y la *selección*

Completamos la serie de vida artificial con un artículo que describe el fenómeno de la evolución desde las perspectivas biológica y computacional, describiendo cómo utilizar la evolución en nuestros programas para resolver problemas y crear sistemas autoconfigurables.

Figura 1: Zooland es una buena guía de recursos de Vida Artificial.



artificial que nosotros establecemos en nuestras granjas o en nuestros programas, es que la selección natural no es propiamente una selección [C7]. Nosotros podemos seleccionar para la reproducción los seres que más nos interesan, ya sean las vacas más lecheras o los *agentes* software que mejor resuelven un problema. En cambio, en la naturaleza no existe -en principio- una inteligencia exterior que determine la dirección de la evolución. Y sin embargo la evolución sí se produce. Si efectivamente, no existe *algo* o *alguien* que controle la evolución de la vida en nuestro planeta, entonces nosotros mismos seríamos el ejemplo de un principio básico y universal: que la vida, la inteligencia, la consciencia y quién sabe qué otros tipos de complejidad en el futuro, son sucesos inherentes, inevitables y espontáneos de nuestro universo. A esto también se le ha llamado *Darwinismo Universal* [C1], y supone que toda vida, en cualquier lugar, habría evolucionado por medios darwinianos.

El origen de la vida y el programa Tierra

El programa *Tierra* [A1] es un ejemplo de cómo agentes software pueden evolucionar sin que sea necesaria una selección dirigida por una entidad externa. Tanto en este programa como en otros, existe una serie de componentes software de algún tipo capaces de reproducirse y sufrir mutaciones. En un *Algoritmo Genético*, los agentes deben resolver un problema particular, pero en *Tierra* los agentes no hacen nada más que reproducirse. El espacio de memoria limitado produce una *selección natural*, ya que sólo las mejores entidades podrán dejar descendencia en él. La existencia de pequeñas mutaciones aleatorias basta para generar agentes con características muy complejas, capaces de invadir o cooperar con otros agentes. Después de estudiar una de estas simulaciones, parece lógico suponer que en nuestro planeta haya podido suceder algo parecido. Está bastante extendida la idea de que las primeras entidades replicantes surgieron al *azar*, a partir de la combinación de elementos, y que la selección hizo el resto. Sin embargo Thomas Ray decidió que su programa comenzara con un primer agente capaz de copiarse a sí mismo, sin pretender que esta autocopia se produjera espontáneamente, como hizo Steen Rasmussen. Se plantea la cuestión de sí la probabilidad de aparición

Figura 2: Para los aficionados a Linux y a la Vida Artificial, este es un sitio imprescindible.

de los componentes básicos de la vida es demasiado baja para un sólo planeta. Fred Hoyle [C4] sugiere la existencia de un bombardeo de material genético del exterior, ya sea con o sin entidad consciente detrás de ello, lo que daría un mayor margen al haber podido surgir este "primer replicante" en cualquier otro mundo. Además, esto solucionaría algunos otros problemas, como los aparentes "saltos de complejidad" evolutivos. Es sorprendente la aparición de órganos complejos como los ojos, que difícilmente han podido surgir de una evolución gradual si no proporcionan ninguna ventaja al individuo hasta que no se encuentran formados por completo. El *lamarkismo*, es decir, la transmisión genética de los caracteres adquiridos, está resucitando[C7]. Máximo Sandín [C9] ofrece otra interesante explicación a todas estas cuestiones mediante infecciones de tipo vírico capaces de afectar rápidamente a gran parte de la población.

La selección natural no es propiamente una selección

En mi opinión, lo más apasionante de las difícilmente explicadas por el darwinismo es la cuestión de la aparición de sentimientos de placer y dolor en los seres vivos. Nosotros podemos asignar a cada agente una variable con un número llamado placer o dolor, pero no es necesario que la entidad tenga *realmente* esas sensaciones para que se comporte como si las tuviera. Tal vez podamos construir algún día robots que se comporten como seres humanos, pero ¿Podremos hacer que *sientan*? Y aunque pudiésemos, ¿Por qué hacerlo? ¿Por qué lo ha hecho la naturaleza con nosotros? Una cosa es la vida artificial como imitación de los *procesos* propios de la vida, y otra muy distinta es la recreación de su esencia. En fin, las teorías sobre la vida y la evolución son difícilmente demostrables y el debate es intenso. Podría parecer que estas cuestiones sólo atañen a los biólogos y filósofos, pero no es así. Recordemos que se trata de crear inteligencia en nuestros ordenadores imitando la forma en que lo hizo la naturaleza. Sin embargo, no podemos hacerlo sin más. Debemos captar al máximo la esencia de todo el proceso para evitar que nuestro ordenador tarde también millones de años.

¿Evolución egoísta, programas egoístas?

La teoría de la evolución puede llevarse directamente a su extremo lógico en forma de *darwinismo social* suponiendo que *la lucha por la vida y la supervivencia del más apto* son la única vía de todo progreso humano. También se puede tomar la postura de Pedro Kropotkin [C5] y defender el *apoyo mutuo* como otro factor de evolución. En los humanos se encuentran comportamientos muy distintos. Desde cierto punto de vista, se puede argumentar que cuando un individuo coopera, es porque, egoístamente, ha evaluado que esto le ayuda a conseguir sus objetivos, y por tanto sólo existe el egoísmo. Esta aparente paradoja no es más que una confusión de términos. Para tratar la cuestión adecuadamente es preciso no tener en cuenta motivaciones, sino sólo actos. Un comportamiento altruista será aquel que contribuya al bienestar ajeno a expensas del propio; uno egoísta sería exactamente lo contrario [C1]. Voy a llamar *cooperación* a las diferentes formas de acuerdo que se encuentran oscilando en el límite entre altruismo y egoísmo.

Antes de continuar quiero aclarar que no estoy discutiendo la ética de estos comportamientos, sino analizando por qué algunos o una combinación de ellos son seleccionados en la evolución. Para saber cuáles son los seleccionados, basta con mirar a nuestro alrededor, ya que nosotros y el resto de los seres vivos somos producto de ella. Aún así no debemos olvidar que el proceso evolutivo es algo dinámico. Hoy se selecciona una cosa, mañana otra. El hombre seguirá evolucionando, no sabemos hacia dónde. Tal vez sea lo suficientemente diferente como para dar un giro radical al proceso, tal vez no. En mi opinión, y en contra de la de muchos otros, el autor que mejor trata este tema es Richard Dawkins [C1] [C3]. Para él somos *máquinas de supervivencia* construidas por nuestros genes para su propia perpetuación. Venimos de los *genes egoístas*, moléculas autorreplicantes que en cierto momento "decidieron" que la creación de una máquina como nosotros, con una capacidad de razonamiento flexible, era lo más adecuado para sus fines. Hay idealistas que rechazan impulsivamente la idea de una naturaleza basada en *genes egoístas* y hay quienes sólo ven en ella egoísmo y destrucción. Si bien es cierto que demasiadas veces la naturaleza no es *madre*, sino más bien *cruel* suegra, fuera de nuestra óptica, la naturaleza es simplemente indiferente. En cualquier caso, Dawkins habla de *genes* egoístas, no de *individuos* egoístas. Y lo que es más importante: con la teoría de Dawkins, la cooperación e incluso el altruismo (reales) entre individuos pueden ser explicados por el "egoísmo" (metafórico) de los genes.

En la evolución se combinan egoísmo y altruismo

Vamos a verlo. En todas las células de nuestro cuerpo existe la misma información genética: una larga cadena de ADN idéntica en el interior de cada célula. Existen muchas definiciones de *gen*. Para Dawkins, *"el gen egoísta no es sólo una porción física de ADN: es todas las réplicas de una porción de ADN, distribuidas por el mundo"*. Es decir, el mismo *alelo* o valor en las mismas posiciones dentro de la cadena de ADN, es el *mismo* gen egoísta, ya se encuentre en uno o en distintos individuos. Un *gen egoísta* de los

Figura 3: Enfoque subsimbólico de la IA.

- **Inteligencia Artificial**
    - **Enfoque Simbólico o *bottom-up***
    - **Enfoque Subsimbólico o *bottom-up***
        - **Redes Neuronales Artificiales**
        - **Computación Evolutiva o Algoritmos Evolutivos**
            - **Solidificación o Recocido Simulado (*Simulated Annealing*)**
            - **Algoritmos Genéticos**
            - **Estrategias Evolutivas**
            - **Clasificadores Genéticos**
            - **Programación Genética**

que habla Dawkins no sólo está *simultáneamente* en todas las células de nuestro cuerpo. Está simultáneamente en *varios* individuos. Esta abstracción de Dawkins sirve para expresar la siguiente idea: los genes tienen el objetivo de reproducirse a sí mismos, a costa de lo que sea, pero no a costa de otro segmento de ADN idéntico, ya que ambos son el mismo gen. Podemos decir que las células de nuestro cuerpo cooperan, e incluso que se comportan de forma altruista, ya que no intentan reproducirse a costa de sus vecinas, sino que producen un crecimiento ordenado, formando un cuerpo, tal como conviene a los *genes egoístas*. A nivel de individuo, cuando un individuo coopera con otro y ambos comparten genes (y dos miembros de la misma especie suelen compartir más del 90% de ellos), podemos decir que en realidad lo que ocurre es que los *genes egoístas* se están ayudando a sí mismos. Cuando dos individuos de la misma especie compiten, lo hacen para propagar su 10% diferencial. Es lógico que si dos individuos comparten el mismo *nicho* (por ejemplo, por ser de la misma especie), existirá una mayor competencia entre ellos. Pero ante las mismas circunstancias, los individuos cooperarán con otros en función del número de genes en que coincidan. El apoyo mutuo entre los individuos es, efectivamente, *un factor de evolución* [C5], pero está basado en el egoísmo de los genes.

Genes y músicos de rock

Dawkins ha sido acusado de tratar los genes como unidad de selección [C8] [C9]. Aunque la selección actúa sobre individuos, es evidente que la selección de individuos modifica el conjunto genético de la población, lo que indirecta y estadísticamente produce una selección de genes. ¡También podríamos criticar a casi todos los biólogos por tratar a los individuos como unidad de reproducción! Puede parecer asombroso, pero los animales no nos reproducimos, al menos, no directamente. No producimos copias de nosotros mismos: producimos copias de nuestros genes y éstos, combinados tal vez con los de otro individuo, y afectados por el entorno, producirán un nuevo ser. La cuestión crucial es hasta qué punto la selección de individuos afecta a la selección de genes. Usando una metáfora, los integrantes de un grupo de rock (genes) forman un conjunto musical (cadena de ADN, genotipo) que se desarrolla componiendo canciones (fenotipo, ser vivo) y que se escuchan en la radio (entorno) junto con otras canciones. La selección actúa sobre las canciones (fenotipo), pero es de suponer que la selección de canciones produce, en definitiva, una selección de músicos (genes).

Buenos, pero no tontos

Para Fernando Savater, aspectos éticos como el respeto hacia los demás son actitudes cuyo origen es en última instancia la búsqueda inteligente del beneficio propio [E7]. Las simulaciones por ordenador parecen darle la razón. En juegos como *El dilema del prisionero* [A6] [B8], se observa que el altruismo es perjudicial para el que convive con individuos egoístas, pero el egoísmo necesita a quien explotar a largo plazo, por lo que ambas son estrategias destinadas a desaparecer. Son los pactos propios de la cooperación los que ofrecen los mejores resultados. La ética y las leyes son en realidad manifestaciones de estos *pactos de cooperación* que la evolución selecciona como útiles para nuestra propia supervivencia.

Computación Evolutiva

Espero haber presentado una visión aceptable del mecanismo de la evolución. Es el momento de pasar estas ideas a nuestros programas. ¿Por donde empezar? ¿Qué tipos de *Computación Evolutiva* existen? Las nuevas ciencias siempre pasan por una fase inicial de desconcierto e inconsistencia hasta llegar a unas convenciones aceptadas por todos. Más o menos la clasificación aceptada en nuestros días es la de la figura 3.

El enfoque subsimbólico de la IA se caracteriza por crear sistemas con capacidad de aprendizaje. Éste se puede obtener a nivel de individuo imitando el cerebro (*Redes Neuronales*), o a nivel de especie, imitando la evolución, lo que se ha denominado *Computación Evolutiva* (CE), término relativamente nuevo que intenta agrupar un batiburrillo de paradigmas muy relacionados cuyas competencias no están aún muy definidas. Hasta hace poco era común hablar de *Algoritmos Genéticos* (AG) en general, en

vez de identificar diferentes tipos de CE, ya que el resto de los algoritmos se pueden interpretar como variaciones o mejoras de los AG, más conocidos. En un AG los elementos de las cadenas (genes) son típicamente bits, y en ciertos casos, valores numéricos o strings. Por su parte, en las *Estrategias Evolutivas* los elementos de las cadenas son reglas que definen cómo va a actuar el individuo ante cierta situación, y en la *Programación Genética* son instrucciones en un lenguaje de programación. *Simulated Annealing* se puede considerar una simplificación de los AG cuyo origen está en los procedimientos físicos de solidificación controlada. Los *Clasificadores Genéticos* solucionan problemas de reconocimiento de patrones mediante un entrenamiento basado en ejemplos, almacenando en las cadenas información que relacione los datos de entrada y la salida deseada. Finalmente, a mí me gusta considerar la *Vida Artificial* como un superconjunto de todas estas técnicas [B11].

Opciones de un Algoritmo Evolutivo

Aunque existen otros esquemas [B6], por lo general, estos programas comienzan con una fase de inicialización de las entidades y su entorno, y seguidamente ejecutan repetidamente ciclos dentro de los cuales podemos distinguir tres etapas (fig. 4):

Figura 4: Etapas de algoritmo evolutivo.

- **Evaluación**: se trata de asignar un valor de peso o *fitness* a cada individuo, en función de lo bien que resuelve el problema.
- **Selección**: ahora debemos clasificar a los agentes en cuatro tipos, según sobrevivan o no, y según se reproduzcan o no, en función de los pesos.
- **Reproducción**: se generan los nuevos individuos, produciéndose algunas mutaciones en los nacimientos.

En realidad las dos primeras fases se pueden fundir en una, ya que no es estrictamente necesario asignar a cada entidad un peso, sino simplemente saber cuáles son mejores que otras, pero asignar pesos suele ser lo más cómodo. También podemos combinar la segunda y la tercera generando las nuevas entidades únicamente en la zona de memoria donde se encuentran los agentes a eliminar, y mantener así la población constante. En cualquier caso, siempre existirá alguna forma de *evaluación*, *selección* y *reproducción*. Cada uno de estos procesos se puede realizar de muchas formas distintas, independientemente del problema que se esté resolviendo.

En la figura 5 se resumen algunas opciones de un algoritmo evolutivo. Podemos idear muchas más. En general se trata de distintas formas de producir, o bien una convergencia más rápida hacia una solución, o bien una exploración más a fondo del espacio de búsqueda. Ambas cosas son deseables y contradictorias, por lo que se ha de llegar a un compromiso. Este compromiso es lo que antes he llamado cooperación. Por supuesto que todo esto no son más que metáforas. Se requiere de una fuerza conservadora (explotación-egoísmo), que beneficie a los mejores agentes, es decir, a los que mejor resuelvan nuestro problema. Esto es evi-

Figura 5: Algunas opciones de un algoritmo evolutivo.

Opciones Generales
- Número de entidades.
- Número de elementos (genes, reglas) por cada agente.

Método de Evaluación: Asignar un peso
- Desordenar las entidades antes de evaluarlas.
- Diferentes formas de modificación de los pesos después de la evaluación. Por ejemplo, el peso de una entidad se puede calcular independientemente de las demás entidades, o se puede modificar posteriormente este valor, disminuyendo el peso si existe otra entidad muy parecida, analizando para ello un cierto subconjunto de la población vecina.

Método de Selección: ¿Quién muere? ¿Quién se reproduce?
- Con o sin reemplazamiento.
- Método de la ruleta.
- Método de los torneos.
- Seleccionar el n% mejor y el m% peor.

Método de Reproducción: Generar y mutar nuevos hijos
- Los padres pueden tomarse por parejas o en grupos más numerosos, elegidos al azar o en orden de pesos.
- En el caso de detectar que los progenitores son muy parecidos, se puede realizar una acción especial, como incrementar la probabilidad de mutación.
- Las entidades pueden comunicar a otras su conocimiento, ya sea a toda o a una parte de la población, directamente o a través de una *pizarra*, (una especie de tablón de anuncios).
- Método de recombinación de genes: se puede tomar genes de uno u otro progenitor al azar, en un cierto orden, con uno, dos o más puntos de corte, etc.
- Tasa de mutación variable.
- Fijar una tasa de mutación diferente para cada individuo o incluso para cada gen.
- Hacer que sea más probable que se produzca una mutación en un gen si en su vecino ya se ha producido [B4].
- Sustituir por mutaciones genes sin utilidad, como reglas incorrectas o repetidas.
- Tipos de mutaciones.

Figura 6: Descripción de los tipos de opciones en un algoritmo evolutivo.

Tipo gen
Los aspectos de *tipo gen* serán aquellos que, como los genes, son diferentes dentro de un mismo individuo.

Tipo individuo
Los aspectos de *tipo individuo* son aquellos que se mantienen iguales en un mismo individuo, pero difieren de unos agentes a otros.

Tipo isla
Los aspectos de *tipo isla* son aquellos que son iguales para un subconjunto de individuos; de más de uno, pero sin llegar al total.

Tipo población
Los aspectos de *tipo población* son los que deben ser iguales para toda la población.

dente, pero no basta. También es necesaria una fuerza innovadora (exploración-altruismo), que permita la existencia de agentes muy distintos, aún cuando su peso sea menor. Así se puede obtener la variedad suficiente para evitar una población estancada en un máximo local, y permitir la resolución de problemas cambiantes o con varios máximos. Esto ocurre de forma espontánea en la naturaleza por ser algo inherente a cada entidad, que puede ser seleccionado, pero resulta más cómodo programarlo de forma externa, es decir, haremos que nuestro programa seleccione para la reproducción "los agentes buenos", pero también "unos cuantos de los malos" para mantener la variedad.

Los *tipos de mutaciones* requieren de una mayor explicación. Cuando los genes son bits, una mutación consiste en invertir un bit, pero ¿cómo mutar genes cuando cada gen puede tomar mas de dos valores? Supongamos que tenemos un problema con tres variables de 8 bits cada una. Tendríamos cadenas como:

10001000-0101011111-11101110

Vamos a suponer que es probable que existan *zonas de valores* buenas y malas; es decir, que si el 7 es un buen valor y el 200 malo, será probable que también el 8 sea bueno y el 199 malo. Sería interesante que las mutaciones nos permitiesen movernos (cambiar de valor) lentamente y con exactitud si estamos en la zona buena, pero también rápida y bruscamente por si acaso nos encontráramos en la mala. Esto lo podemos conseguir precisamente cambiando un bit al azar, ya que nos movemos lentamente cambiando bits de la parte derecha, y rápidamente con los de la izquierda. En cambio, si la mutación consistiera en cambiar el valor de toda una variable por otro cualquiera, teniendo todos la misma probabilidad, podremos movernos rápidamente, pero no lentamente (al menos, con una probabilidad baja), con lo que perderíamos valores interesantes. Si las mutaciones consistieran en incrementar o restar uno de forma circular, podríamos movernos lentamente, pero no rápidamente, con lo que costaría mucho salir de la zona mala. En definitiva se trata de, dado un valor, definir la probabilidad de pasar a cada uno de los valores restantes.

Hemos supuesto que existen *zonas de valores* buenas y malas. Si esto no se cumpliera podríamos simplemente "incrementar uno" de forma circular, para recorrer todos los valores cuanto antes. En general se observa que las mutaciones a nivel de bit son las más adecuadas, aunque no siempre ocurre así. En cualquier caso, cuando combinemos dos cadenas deberemos actuar a

Figura 7: Distintas formas de incluir tasas de mutación variables.

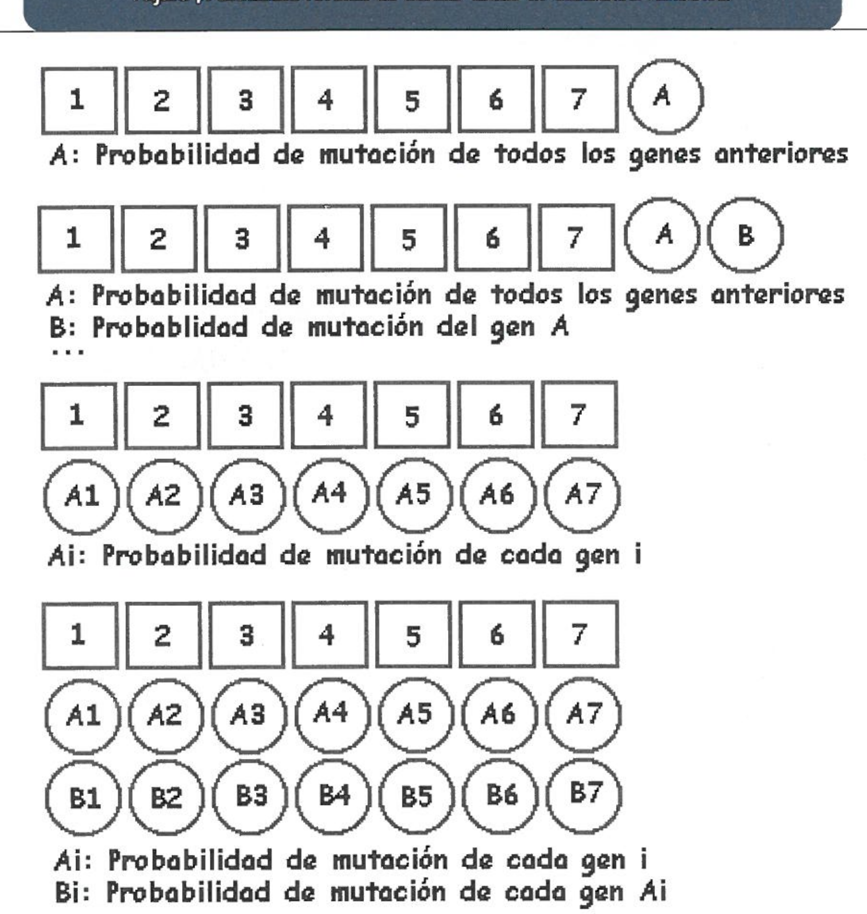


Figura 8: Algoritmos genéticos jerárquicos en tres niveles.

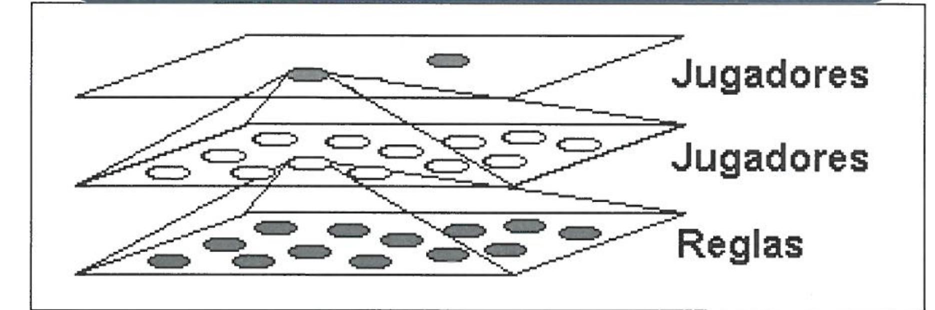
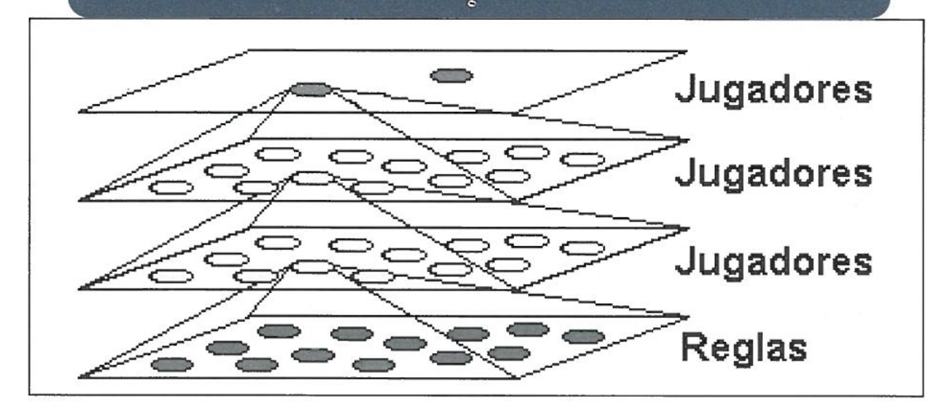


Figura 9: El esquema admite múltiples niveles intermedios.



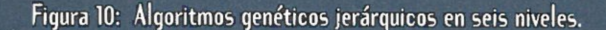
nivel de variable, y combinar variables completas. De lo contrario estaríamos generando multitud de mutaciones simultáneas, rompiendo por cualquier lugar valores que pueden ser valiosos.

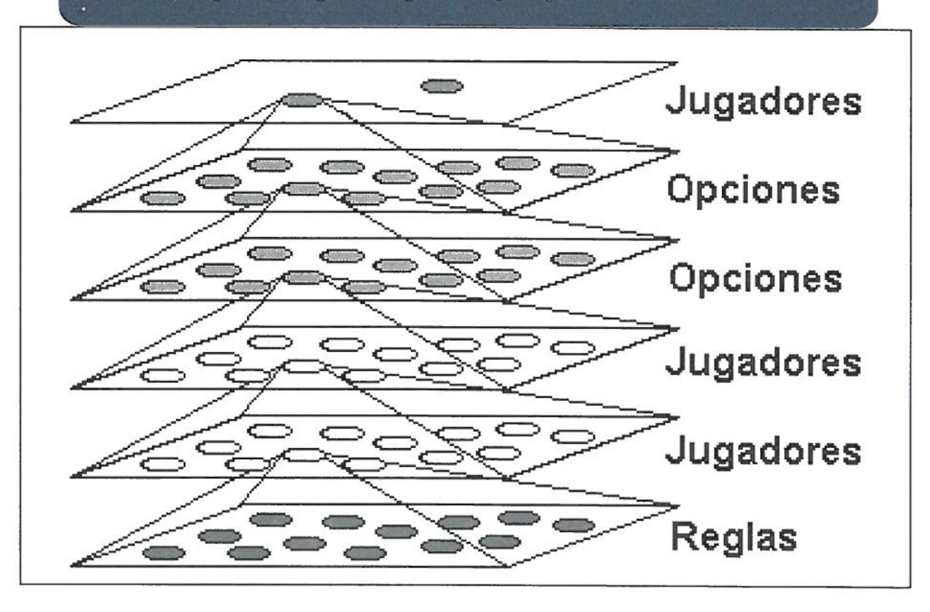
Programas autoconfigurables

¿Qué opciones son las buenas? Tal vez sea posible determinar a simple vista si una elección es adecuada, pero en la mayoría de los casos será necesario un tedioso proceso de prueba, ya que la bondad de unas puede depender de la elección de otras. Si algo se aprende programando algoritmos evolutivos es que las *ideas geniales* no siempre lo son. Es muy corriente la existencia de parámetros que aunque disminuyen el número de ciclos necesarios para llegar al tipo de entidad que buscamos, en la práctica no son rentables dado el incremento de la duración de los ciclos.

Ejecutar todas las posibles combinaciones de opciones es inconcebible ¿No será posible analizar varias en un único algoritmo, en diferentes momentos? ¿Y que tal analizar varias opciones *simultáneamente*, en fracciones separadas de la población? Es decir, ¿Podemos crear algoritmos evolutivos autoconfigurables? Para responder a estas preguntas es interesante agrupar todos los aspectos y opciones en cuatro categorías (figura 6).

Las opciones o **aspectos de tipo gen** pueden probarse introduciéndolos como parte de cada uno de los genes, *dentro* de cada uno de los genes. Si se quiere, se puede pensar en una matriz de dos dimensiones más que en una cadena de genes. En el caso de un programa que aprende a jugar al tres en raya [B6] como el que aparece en el CD-ROM. Cada uno de los genes es una regla que dice cómo jugar ante determinada situación. Antes de realizar un movimiento, el agente comprueba qué reglas puede usar y elige una de ellas. ¿Cómo elegirla? Podríamos pensar en asociar inicialmente a cada regla una prioridad arbitraria fija, y elegir la de mayor prioridad. Otra forma sería asignar a cada regla un peso que se incrementase cada vez que dicha regla fuera usada en una partida ganada, y aplicar la regla que en más victorias haya participado hasta el momento. Para llevar a la practica cualquiera o las dos opciones combinadas, bastará con incluir la información como parte de cada gen. Si antes cada


Figura 10: Algoritmos genéticos jerárquicos en seis niveles.



parte de cada gen. Si antes cada gen era una regla, ahora cada gen será una regla, más un valor de prioridad, más un peso. Otra opción de *tipo gen* sería asignar una probabilidad de mutación distinta para cada elemento de la cadena, de forma que los genes, después de ser heredados, tengan diferentes probabilidades de mutarse [B12] [B13] [B14] (ver fig. 7). Si en un determinado momento, cierto valor de un gen se muestra como muy útil, puede ser interesante que su probabilidad de mutación disminuya, y viceversa.

Los **aspectos de tipo individuo** pueden probarse introduciéndolos como si fueran nuevos genes, a continuación de los que ya existen. La tasa de mutación, aunque suele ser igual para toda la población, y hemos visto que podría ser incluso diferente para cada gen, podría también ser distinta para cada individuo, de forma que los hijos de ciertos agentes tengan mayor probabilidad de poseer mutaciones. Por ejemplo, en el tres en raya, una entidad estará compuesta por un conjunto de reglas más una probabilidad de mutación. Este valor se podrá heredar y mutar, y si es útil, se verá seleccionado, igual que una regla cualquiera. Antes hemos hablado de asignar una prioridad a cada regla. La prioridad asociada a cada regla es un aspecto de *tipo gen*, pero la decisión de asociar o no prioridades a las reglas es un aspecto de *tipo individuo*, que debe afectar obligatoriamente a todas sus reglas.

Probar e implementar **aspectos de tipo isla** va a ser más complicado. Por ejemplo, supongamos que queremos comprobar que ocurriría si permitimos a nuestros bichos compartir su conocimiento, es decir, transmitirse información genética en forma de infección vírica, tal como relata Máximo Sandín [C9]. Podríamos crear varias *islas* o sub-grupos de agentes con distintas opciones, manteniéndolos separados por una *barrera* de algún tipo que no permita el contacto de unos con otros. Más tarde podríamos comparar el conocimiento obtenido en cada *isla* o incluso juntar de vez en cuando a algunos o a los mejores de cada grupo para la reproducción [B2]. Las opciones de *tipo isla* pueden ser evaluadas automáticamente sin modificar mucho el algoritmo, ya que es de esperar que los grupos con peores opciones se extingan. La idea general es que tanto para grupos como para individuos o genes, lo bueno sobrevive y lo malo perece. Para crear barreras, podemos definir tipos de especies distintas que sólo son capaces de reproducirse entre sí, poseyendo cada una de ellas, por ejemplo, distintos métodos de recombinación de genes, con uno, dos o más puntos de corte en el cruzamiento, o incluso concibiendo de forma distinta los puntos de comienzo y fin de las unidades mínimas que se recombinan (genes). Curiosamente, en este caso, el propio método de implementación de las barreras (tipos de reproducción incompatibles) es precisamente aquello que estamos probando. Para implementarlo podemos añadir un gen especial a cada agente de forma que sólo los agentes con el mismo valor en dicho gen sean capaces de recombinarse entre sí. Cada valor corresponderá con una opción diferente. Para probar más de una deberemos poner varios de estos genes, y obligar a una coincidencia en todos ellos.

Las **opciones de tipo población** sólo pueden probarse ejecutando algoritmos en paralelo. Estas opciones son las más generales, como el número total de agentes o el número de reglas que tiene cada individuo. Otro ejemplo de *tipo población* es el hecho de *ajustar* los pesos de los agentes, de forma que si ya existe otro agente parecido al que acaba de nacer, se disminuye el peso del nuevo agente. Esta opción no puede ser aplicada sólo a una parte la población, ya que

Figura 11: Referencias sobre vida artificial.

[A1] *"Jugué a ser Dios y creé la vida en mi computadora"*. Ray, Thomas S. Emocionante relato que describe una de las primeras simulaciones de vida por ordenador. http://www.hip.atr.co.jp/~ray/pubs/spanish/spanish.html

[A2] *"Vida Artificial"*. Prata, Stephen. 1993. Ed. Anaya. Buen libro para iniciarse en el tema.

[A3] Swarm: multi-agent simulation of complex systems. Plataforma de desarrollo de vida artificial para Unix Linux GNU. http://www.santafe.edu/projects/swarm/

[A4] Vida Artificial en español. J. J. Merelo. Universidad de Granada. http://kal-el.ugr.es/VidArt/VidaArti.html

[A5] Linux Alife. Vida Artificial y recursos para Linux. http://aisun0.ai.uga.edu/~jae/ai.html

[A6] *"Vamos a Crear Vida en un PC"*. Herrán, Manuel de la. Revista Solo Programadores nº 36 Agosto 1997.

[A7] *"Emergent Computation: Self Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks"*. Forrest, Stephanie (Ed.) Revista Artificial Intelligence nº 60. 1993.

[A8] *"Agent Oriented Programming"*. Shoham, Yoar. Revista Artificial Intelligence nº 60. 1993.

[A9] *"Vida Artificial"*. Fernandez Ostolaza, Julio, y Moreno Bergareche, Alvaro. 1992. Ed. Eudema. Epistemología de la Vida Artificial, pasa revista a la historia de la Vida Artificial.

[A10] Gaia. Algoritmos Genéticos, Vida Artificial y Aprendizaje. Proyecto de una plataforma de Vida Artificial. http://www.geocities.com/SiliconValley/Vista/7491/

Figura 12: Referencias sobre Computación Evolutiva y aprendizaje.

[B1] *"Algoritmos Genéticos"*. Holland, John H. Revista Investigación y Ciencia, Septiembre 1992. Fantástico resumen del "padre" de los algoritmos genéticos.

[B2] Matemáticas Aplicadas: Vida Artificial y Algoritmos Genéticos. Blanca Cases y Pedro Larrañaga. Universidad del País Vasco. Muy interesante. http://www.geocities.com/CapeCanaveral/9802/

[B3] *"Sistemas expertos en contabilidad y administración de empresas"*. Sierra Molina, Guillermo, y otros. 1995. Editorial Ra-Ma. Algorimos Genéticos, Redes Neuronales y otras técnicas de resolución de problemas, explicado de una forma clara y sencilla.

[B4] *"Algoritmos Genéticos Avanzados"*. Herrán, Manuel de la. Revista Solo Programadores nº 37 Septiembre 1997.

[B5] *"Made Up Minds"*. Drescher, Gary L. 1991. MIT Press. Fantástico libro sobre el aprendizaje automático.

[B6] *"Aprendizaje Automático y Vida Artificial"*. Herrán, Manuel de la. Revista Solo Programadores nº 38 Octubre 1997.

[B7] *"Adpatation in Natural and Artificial Systems"*. Holland, John H. University of Michigan Press. 1975.

[B8] *"Temas Metamágicos"*. Hofstader, Douglas R. Revista Investigación y ciencia, Agosto 1983.

[B9] *"Agentes Autodidactas ¿Futuro o Realidad?"*. Herrán, Manuel de la. Revista BASE nº 30 Noviembre 1996.

[B10] *"El ordenador cerebral"*. Jáuregui, José Antonio. 1990. Editorial Labor. Un punto de vista muy original acerca del ser humano, la inteligencia y el darwinismo.

[B11] *Guía del Autoestopista de la Computación Evolutiva ("The Hitch-Hiker's Guide to Evolutionary Computation")* Heitkötter, Jörg and Beasley, David, eds. FAQ de comp.ai.genetic con definiciones y clasificaciones de los términos más comunes en Computación Evolutiva. http://www.etsimo.uniovi.es/ftp/pub/EC/FAQ/www/top.htm

[B12] *"J. Theo. Biology"*, vol. 17. Reed et al. 1967.

[B13] Ph.D. dissertation, U.Mich. Rosenberg. 1967.

[B14] *"Handbook of Evolutionary Computation"*. Baeck, Thomas; Fogel, David B. (eds.) 1997. Oxford, NY.

[B15] *"Evolutionary Computation: Toward a New Philosophy of Machine Intelligence"*. Fogel, David B. 1995. IEEE Press.

se vería en obvia desventaja. Evaluar opciones de *tipo población* es lo más complicado. Deberemos crear algoritmos independientes y compararlos, por ejemplo, en el tres en raya, provocando una partida entre las mejores entidades de cada una de las simulaciones, es decir, creando otro *plano* superior de juego.

Algoritmos jerárquicos

Voy a cambiar de punto de vista para analizar las jerarquías dentro de los algoritmos evolutivos. En un algoritmo genético como el de las palabras y frases del CD-ROM [B4], la reproducción, selección, mutaciones, asignación de pesos, etc. se desarrollan a un sólo nivel o *plano*. Sin embargo, en el caso del tres en raya [B6], existen varios *planos*.

En el plano más elevado, juegan el ordenador contra el hombre (Ver fig. 8). Este juego puede interpretarse como un algoritmo evolutivo reducido a su mínima expresión, con una población de dos individuos, donde el algoritmo termina al finalizar la partida. En este tipo de problemas tienen bastante éxito los paradigmas en los que el ordenador trata de aprender observando el juego humano, pero no es este el caso. Para aprender, el ordenador crea un segundo nivel de juego, donde muchas entidades jugarán por parejas. Aquella entidad que más partidas gane será la que represente al ordenador en el juego contra el hombre. Pero todavía se puede hablar de un tercer nivel, ya que dentro de un mismo agente, cada una de las reglas puede poseer también un peso, y aunque las reglas no se reproducen combinando sus elementos (podrían hacerlo), es posible sustituir algunas cada cierto tiempo.

Ya que la entidad de nivel superior ha creado un sub-mundo o plano donde juegan varias sub-entidades, gracias a las cuales es capaz de aprender por su cuenta antes de jugar contra el hombre, estas sub-entidades podrían a su vez crear otros planos que les proporcionasen conocimiento, hasta que llegue el momento de jugar "de verdad". Es decir, una entidad, al igual que jugar, o reproducirse, podría también tener una acción que podríamos llamar "pensar" y que consistiera en experimentar partidas hipotéticas (hipotéticas para la entidad superior; a la inferior le parecerán las suyas tan reales como a cualquier otra), creando "mundos virtuales", de la misma forma que las personas imaginamos o creamos hipóte-

Figura 13: Referencias sobre darwinismo, genética, y evolución.

[C1] *"El gen egoísta"*. Dawkins, Richard. 1994. Salvat Ciencia.

[C2] Introducción en *"El Origen de las especies"* de Charles Darwin. Leakey, Richard E.1994. Ediciones del Serbal - Reseña. Exposición sencilla y muy acertada del darwinismo.

[C3] *"¿Tiene sentido la vida fuera de sí misma?"*. Dawkins, Richard. Revista Investigación y ciencia, Enero 1996.

[C4] *"El universo inteligente"*. Hoyle, Fred. 1983. Ed. Grijalbo. Una valiente crítica al darwinismo.

[C5] *"El apoyo mutuo"*. Kropotkin, Pedro. 1970. Ediciones Madre Terra. Trata el darwinismo y la cooperación.

[C6] *"La evolución sin Darwin: La biología ultramontana"*, en la revista "Revista de Libros", número 9 (septiembre de 1997). Castrodeza, Carlos. 1997. Ed. fundación Caja Madrid.

[C7] *"Un siglo después de Darwin 1. La Evolución"*. Barnett y Otros. 1962. Alianza Editorial.

[C8] *"El pulgar del panda"* Gould, Stephen Jay. Ed. Hermann Blume.

[C9] *"Lamarck y los mensajeros. La función de los virus en la evolución"*. Sandín, Máximo. 1995. Ed. Istmo.

sis para prever los acontecimientos. En la figura 9 se ilustra esta posibilidad.

Este sería además un esquema donde implementar el algoritmo *minimax* típico de los juegos, donde el agente evalúa posibles movimientos propios y del contrario a varios niveles de profundidad. Podemos combinar este tipo de jerarquías con las relativas a la elección de **opciones de tipo población.** Es decir, mientras buscamos la solución a nuestro problema inicial en los algoritmos o planos inferiores, podemos experimentar en un plano superior algunas opciones, las cuales pueden a su vez tener subopciones, tal como ilustra la figura 10.

El problema más difícil

Una cadena de un algoritmo evolutivo tiene varios elementos (genes) que corresponden con variables del problema a resolver. Puede haber genes cuyos valores sean intrínsecamente buenos o malos, es decir, buenos o malos independientemente de los valores del resto de la cadena. Vamos a darles un nombre a este tipo de genes, por ejemplo, genes *no-vinculados*. Consecuentemente, un gen cuya bondad o maldad dependa de los valores de algunos de los otros genes será *vinculado*.

La propiedad de ser o no *vinculado* puede aplicarse también a un conjunto de elementos de la cadena, si tienen en cuenta todos ellos juntos, como un único gen, sin requerir que sean contiguos. Imaginemos un problema en el que todos los genes fueran *no-vinculados*: ¡No sería necesario un algoritmo evolutivo! Para encontrar los valores óptimos de las variables, bastaría con analizar independientemente todos los posibles valores de cada gen, manteniendo constante el resto de la cadena con cualquier valor. Imaginemos ahora que todos los segmentos son vinculados, menos dos, es decir, todos los genes son intrínsecamente buenos o malos, excepto dos, que dependen de otros. Sí, pero ¿de cuáles? ¿El uno del otro? ¿Ambos de todos los restantes? Si la bondad de cada uno de estos dos segmentos sólo dependiera de sí mismo y del valor del otro podrían ser tratados en conjunto como un único gen

Figura 14: Referencias sobre Redes Neuronales.

[D1] Capítulo "Perceptrones" en Dewney, A. K. *"Aventuras Informáticas"*. Ed Labor, 1990. Los perceptrones corresponden con el nacimiento de las redes neuronales, y suponen una versión simplificada de éstas.

[D2] Capítulo "Redes Neuronales" en Sierra Molina, Guillermo, y otros. *"Sistemas expertos en contabilidad y administración de empresas"*. Ed. Ra-Ma, 1995. Explica el funcionamiento más básico de redes neuronales.

[D3] *"Redes Neuronales"*. Luna, Raúl. Solo Programadores 1994.

[D4] *"Redes Neuronales"*. Obiol, Pablo. Jumping. Julio 1997.

[D5] Capítulos "Aprendizaje y Generalización" y "Redes Borrosas y Evolutivas". En Ignacio Olmeda y S. Barba. *"Redes Neuronales Artificiales: fundamentos y aplicaciones"*. 1993.

[D6] *"Chaos, artificial neural networks and the predictability of stock market returns"*. Olmeda, Ignacio y Pérez, Joaquín. 1994.

[D7] *"Neural networks forecasts of intra-day futures and cash returns"*. Isasi, Pedro; Olmeda, Ignacio; Fernández, Eugenio, y Fernández, Camino.

Figura 15: Otras referencias.

[E1] *"Guía del Autoestopista Galáctico"*. Adams, Douglas. 1983. Editorial Anagrama. Ciencia ficción, relato de humor. Presenta el planeta Tierra como un super-ordenador biológico.

[E2] *"Visitantes Milagrosos"*. Watson, Ian. 1987. Ed. grupo zeta. Estados alterados de conciencia, fenómeno ovni, ecología, enseñanzas sufíes, mecánica cuántica y teorema de Gödel. Único.

[E3] *"El misterio de la isla de Tökland"*. Gisbert, Joan Manuel. 1981. Ed. Espasa-Calpe.

[E4] *"Las nueve Revelaciones"*. Redfield, James. 1997. Ediciones B, S.A. Evolución, Teilhard de Chardin y Gestalt.

[E5] *"Las voces del desierto"*. (Muttant Message Down Under). Morgan, Marlo. 1996. Ediciones B.

[E6] *"El problema de la consciencia"*. Chalmers, David J. Revista Investigación y ciencia. Febrero 1996.

[E7] *"Ètica para amador"* Savater, Fernando. 1996. Ed Ariel.

no-vinculado, y aplicar el mismo proceso. Si la bondad de los genes *vinculados* dependiera de todos los demás podremos utilizar el mismo método, teniendo la precaución de calcular primero los valores de los segmentos *no-vinculados*.

En un problema complicado, los valores de los genes no son intrínsecamente buenos ni malos

Imaginemos ahora el caso extremo de que todos los genes sean *vinculados*, y todos dependan del valor de todos los demás. Este no sería un problema irresoluble, pero sí el peor que nos podríamos encontrar, no sólo para las técnicas evolutivas, sino para cualquier otro método. Sería algo así como buscar un objeto con los ojos cerrados, y la mejor forma de tratarlo sería, lógicamente, una búsqueda secuencial. En cualquier caso, lo que más nos interesa es buscar segmentos y grupos de segmentos *no-vinculados*, aunque sean grandes, ya que si podemos dividir la cadena en varios segmentos *no-vinculados* podremos aplicar un algoritmo independiente para cada segmento, en el que el resto de la cadena sea siempre fijo, con un valor cualquiera, y obtener la solución mucho más rápido.

Para saber si un elemento (o un conjunto de elementos) es *no-vinculado* podríamos fijar valores al azar para el resto de la cadena, evaluar las cadenas que resulten de todas las posibles combinaciones de valores para el conjunto de elementos elegido y almacenar el orden de las cadenas ordenadas según su peso. Si repitiendo el proceso utilizando distintos valores en el resto de la cadena, siempre obtenemos las cadenas en el mismo orden, entonces estaremos ante un segmento *no-vinculado*. Otra forma sería comprobar estadísticamente si en distintos ciclos del algoritmo, los mismos valores para un segmento producen posiciones relativas parecidas dentro de la lista de agentes ordenados por peso. Por ejemplo, si una variable puede tomar valores de 1 a 10, podemos almacenar, para cada valor, cuál es la posición media que ocupa la cadena que la contiene, dentro de la lista de cadenas ordenadas por pesos, en cada uno de los ciclos.

Otra forma de encontrar las vinculaciónes sería definirlas al azar y actuar como si de hecho existiesen, comprobando si de esta forma se obtienen mejores resultados. Por último, voy a presentar la que me parece la forma más fácil de todas. Para el algoritmo es mucho más sencillo encontrar los valores óptimos de los segmentos *no-vinculados* que los de los *vinculados*. Por tanto, después de unos cuantos ciclos, los genes de valores más homogéneos tendrán una mayor probabilidad de ser más *no-vinculados* que el resto. Los genes *vinculados* se comportan en realidad como un único gen, por lo que se ha de evitar su división. Se me ocurre que podríamos obtener alguna ventaja si obligamos a que exista una especie de *cohesión* entre los genes *vinculados*, es decir, entre los segmentos de valores más heterogéneos, de forma que se hereden y muten juntos, simultáneamente, con una mayor probabilidad.

Aclaración

La mayoría de las ideas que han aparecido en esta serie de artículos de Vida Artificial se pueden encontrar en la bibliografía y son comúnmente aceptadas, o al menos debatidas. Otras en cambio son *de cosecha propia* tras grandes esfuerzos de reflexión y de programación, y no he podido resistir la tentación de exponerlas. La experiencia me dice que no sería extraño encontrar muchas perfectamente descritas en otros libros que aún no he podido consultar, por lo que quisiera excusarme por anticipado ante los autores no citados. Estos artículos deben leerse con sentido crítico, prueba de ello es que en el nº 37 cometí el fatal error de utilizar incorrectamente el concepto de *no-lineal*. El asunto concreto requiere una explicación demasiado extensa, por lo que no aparece aquí, pero está disponible en: http://www.geocities.com/SiliconValley/Vista/7491/rect_c.htm. Espero al menos haber sido capaz de entretener y crear interés por un tema que tanto me apasiona.

En el CD-ROM se incluye el código fuente completo en Visual Basic 5.0 de los programas mencionados.

Programación de DirectX con Delphi 2.0 (III)

Constantino Sánchez Ballesteros (constantino@redestb.es)

En los artículos anteriores de la serie ya hemos visto la estructura, el funcionamiento y la eficacia del API. Ahora seguimos avanzando en la programación de DirectX con Delphi 2.0 profundizando en los *sprites*, los *pactchings* y la técnica de *clipping*, entre otras.

El concepto "Blitting"

El término *blit* se traduce como transferencia de bloques de *bits*, y es el proceso de transferir o volcar bloques de datos de un lugar de la memoria a otro. Los programadores gráficos utilizan el *blitting* para transferir gráficos de un *buffer* a otro. Los *Blits* son frecuentemente utilizados para crear animaciones de *sprites*, los cuales veremos más adelante.

Para crear *blitting* tenemos varios métodos en DirectX, entre los cuales destacan *Blt* y *BltFast*.

GDI

Los *buffers* que albergan información gráfica son visualizados en la pantalla como imágenes (*bitmaps*, para ser más exactos). En muchos programas Windows, accedemos a esos *buffers* utilizando funciones de Win32 como *GetDC*., que toman el DC (contexto del dispositivo). Después de tener el DC, podemos comenzar a dibujar en los *buffers*. De todas formas, las funciones gráficas de Win32 están diseñadas como una parte diferente del sistema: el *interface* de dispositivos gráficos (GDI). El GDI es un componente que provee abstracción de capas y permite a los programas en Windows dibujar gráficos en la pantalla.

El GDI no fue diseñado para *software* multimedia de alto rendimiento, sino que se creó para aplicaciones de negocios como procesadores de textos y hojas de cálculo. Permite acceso al *buffer* de vídeo en la memoria de sistema, no en la memoria de vídeo, y no toma ventaja de las características especiales que algunas tarjetas de vídeo permiten. En pocas palabras, el GDI es útil para software de negocios, pero su rendimiento es realmente bajo en videojuegos y *software* multimedia.

Por otro lado, DirectDraw puede proporcionarnos *buffers* gráficos que representan la memoria de vídeo actual. Esto significa que cuando usamos DirectDraw, podemos escribir directamente a la memoria de la tarjeta gráfica, activando nuestras rutinas gráficas de forma realmente veloz. Estos *buffers* están representados como bloques continuos de memoria, haciendo fácil el direccionamiento a los mismos.

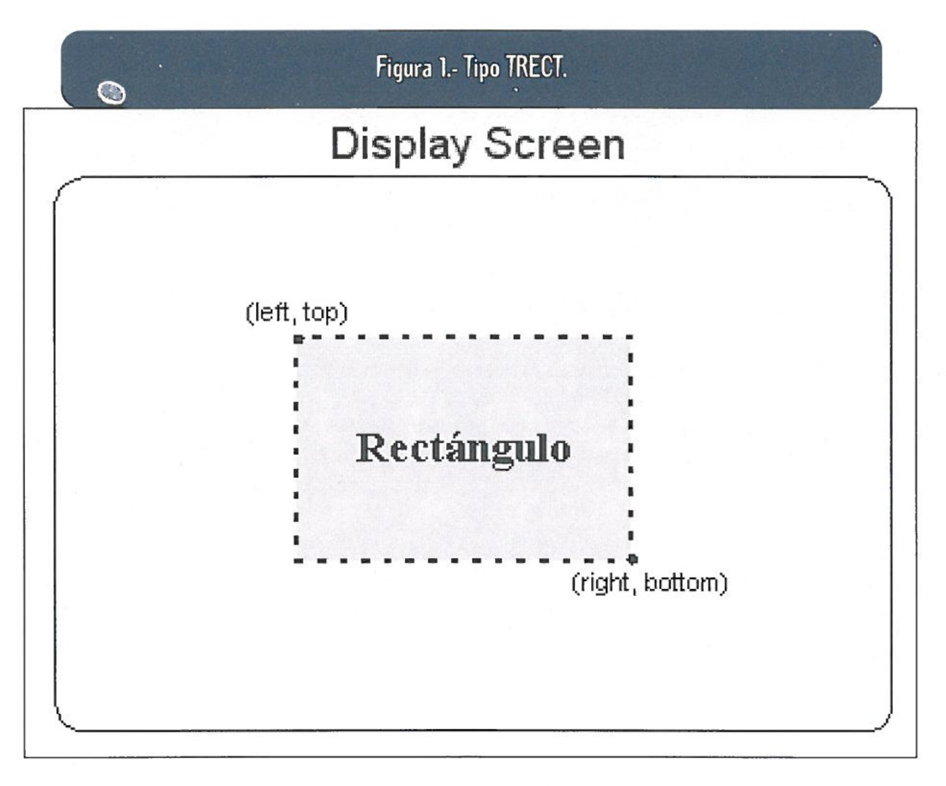
Introducción a los rectángulos

En DirectDraw y la programación Windows, los objetos gráficos representados en la pantalla son referidos en términos de rectángulos. Un rectángulo es descrito por 2 puntos, la esquina superior

izquierda y la esquina inferior derecha. La mayoría de las aplicaciones usan el tipo TRECT en la asignación de coordenadas cuando se efectúan volcados de pantalla entre *buffers* o se detectan colisiones entre objetos que ocupen el mismo espacio. El tipo TRECT presenta la siguiente definición:

En este ejemplo, los miembros *left* y *top* son coordenadas X e Y de la esquina superior izquierda del rectángulo. De forma análoga, los miembros *right* y *bottom* representan las coordenadas de la esquina inferior derecha del rectángulo.

Figura 1.- Tipo TRECT.



Page Flipping y Back Buffering

El *Page flipping* es un *key* en términos multimedia, animación y *software* de juegos. *Page flipping* por *software* es análogo a la forma en que los artistas del *cartoon* (dibujos animados) animan sus dibujos. Por ejemplo, el artista dibuja una figura en una hoja de papel; entonces empieza a trabajar conjuntamente en la segunda hoja. En cada hoja de papel, el artista cambia la figura suavemente para que cuando se pasen las hojas rápidamente aparezca una suave animación.

El *Page flipping* en *software* trabaja de una forma muy similar a este proceso. Inicialmente, creamos una serie de *buffers* que son diseñados para volcarlos en pantalla al igual que el artista pasa de una hoja a otra. El primer *buffer* se refiere al primario y los siguientes se corresponden a los *buffers* secundarios o *back buffers*. Nuestro programa escribe al *back buffer*, entonces vuelca su contenido al primario para que aparezca en la pantalla. Mientras el sistema esté visualizando la imagen, nuestro programa está escribiendo otra vez al *back buffer*. Este proceso se repite tantas veces como se necesite efectuar la animación, permitiendo animar las imágenes de forma rápida y eficiente.

DirectDraw hace fácil el uso del *Page flipping* desde un simple esquema de doble *buffer* (un *buffer* primario con un *back buffer*) hasta esquemas más sofisticados que suman *back buffers* adicionales. En el ejemplo que vimos en el fascículo anterior ya comenzamos a utilizar esta técnica sin haberla mencionado. Esto os puede dar una idea de lo simple que resulta su funcionamiento.

Triple Buffering

En algunos casos, cuando la tarjeta gráfica tiene suficiente memoria, es posible acelerar el proceso de visualización de nuestra aplicación utilizando la técnica del triple *buffer*. El *Triple buffering* utiliza 1 primario y 2 *back buffers*.

No es necesario tener en cuenta todos los *buffers* en una cadena de *triple buffering*. Los únicos que deben referenciarse con punteros son el primario y el secundario. El asignado al primario es necesario para efectuar el volcado de *buffers* de la cadena establecida y el asignado al secundario es primordial para efectuar el *blitting*.

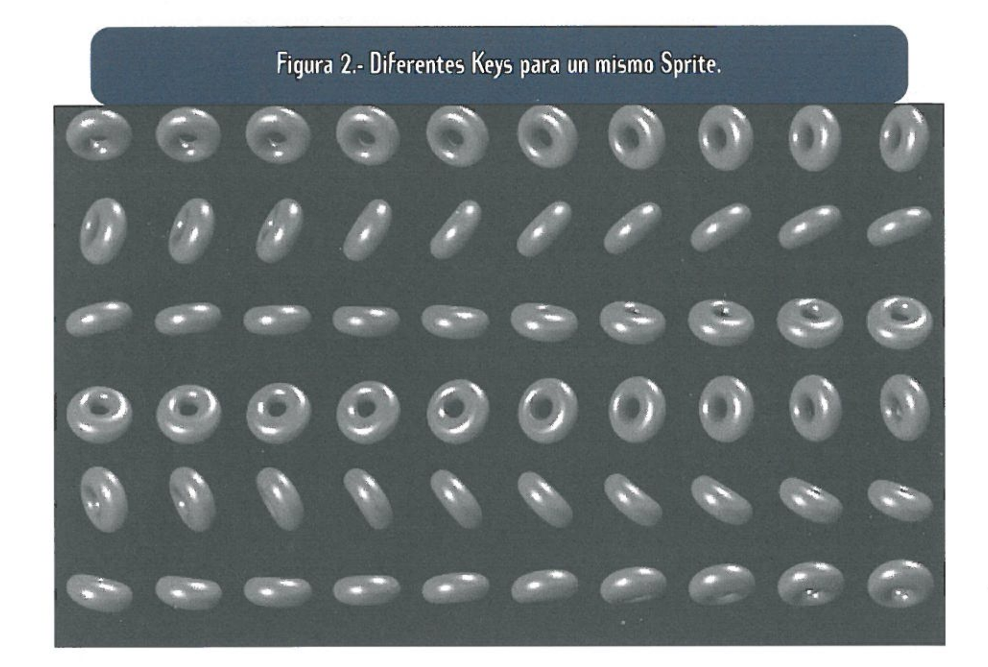
El *Triple buffering* permite continuar el volcado de gráficos al *back buffer* siempre que un método *flip* no se haya completado y el *blit* al *back buffer* haya finalizado. La actuación del método *flip* no se produce como un evento síncrono; Un *flip* puede consumir más tiempo que otro.

Sprites

Probablemente, muchos de vosotros habréis oído alguna vez la palabra *sprite* en el mundillo de los videojuegos. Quizá es una de las partes más importantes en la programación de videojuegos en 2D puesto que nos permiten efectuar animaciones, barras de energía, *scrolls*, etc.

En la práctica, los *sprites* no son más que porciones de mapas de *bits* que están en memoria y son volcados al *buffer* primario para su representación en pantalla. Cuando necesitamos realizar una animación de un personaje, debemos tener en memoria una o varias imágenes representando cada una de las posiciones posibles que el personaje puede realizar para que

Figura 2.- Diferentes Keys para un mismo Sprite.



la animación sea representada en nuestro monitor. El trabajo duro viene dado por el diseño de los personajes, puesto que debemos echar mano de nuestro diseñador gráfico e ir dibujando uno por uno los *keys* (fotogramas) que conformarán la animación.

Los sprites son la parte más importante de la programación gráfica en 2D

Algo relativamente más sencillo es realizar los diferentes *keys* en un programa con soporte 3D como puede ser el 3DStudio, puesto que con tener el modelo necesitado, al rotarlo, transformarlo, etc., obtendremos automáticamente todos sus *keys*. El problema que nos encontramos con este sistema es la complejidad que puede llevar el crear el modelo 3D del personaje. Ya veis que la facilidad de un programa para ejecutar una tarea se torna de forma compleja al realizar otra diferente. Lo ideal es combinar la potencia de los programas 3D con la facilidad de uso de los diseñadores gráficos en 2D.

El concepto técnico del *sprite* a la hora de programar se define como sigue:

- Una vez tenemos albergada la imagen con los diferentes *keys* que conforman la animación, debemos tomar cada uno como si fuese un rectángulo o cuadrado, dependiendo de la longitud del gráfico. Aunque nosotros veamos gráficos con su propio perfil, realmente se están visualizando rectángulos que albergan un gráfico determinado. Para evitar el aspecto cuadrangular de los *sprites* se puede utilizar la técnica de la transparencia, la cual se encarga de que no se dibujen en pantalla los *píxels* que no pertenezcan realmente al *sprite*. Normalmente, los *sprites* albergados en memoria suelen estar dibujados sobre un fondo de color negro. Cuando se toma el rectángulo correspondiente para representarlo en pantalla, la rutina de transparencia se encargará de que no se visualice dicho color, logrando así los perfiles irregulares que representan los *sprites*.

Lo único que debemos tener siempre presente es que trabajaremos con cuatro coordenadas a la hora de visualizarlos, independientemente de la forma y perfil que tengan:

```
(X - Y) (X1 - Y1)
```

X : Representa la coordenada izquierda del Sprite.
Y : Representa la coordenada superior del Sprite.
X1 : Representa la coordenada derecha del Sprite.
Y1: Representa la coordenada inferior del Sprite.

Para efectuar la transparencia en los *sprites* debemos asignar el color que se utilizará como transparente. En DGC tenemos un método mediante el cual asignamos el color deseado para efectuar dicha transparencia:

```
TransparentColor
```

Este método debe asignarse al *buffer* que contiene los gráficos de la siguiente forma:

```
Buffer.transparentColor := 0
```

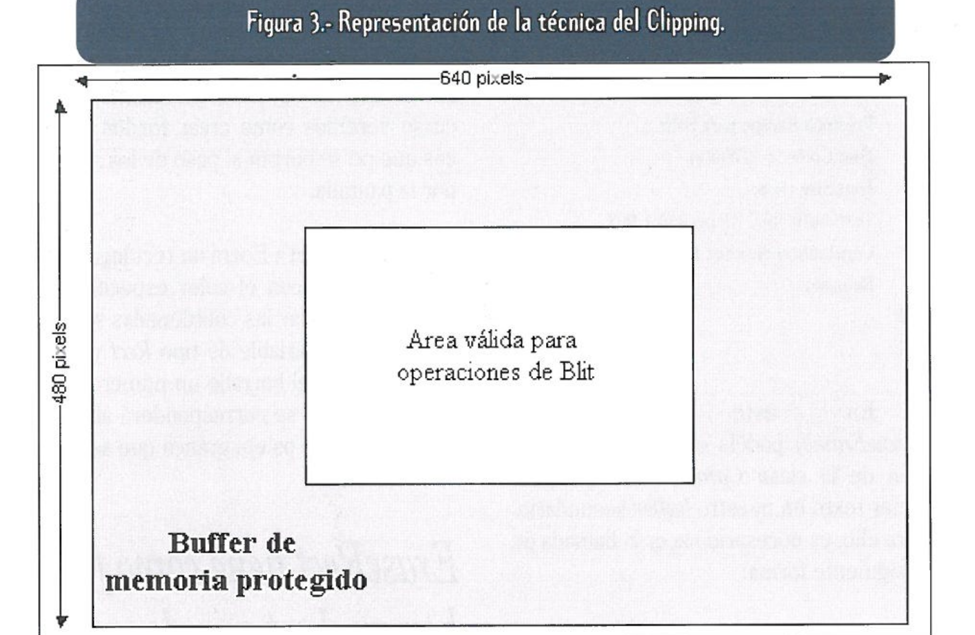
En este caso, se utilizará como transparente el color número 0 correspondiente a la paleta gráfica del *buffer*.

Patching (técnica del parche)

Para completar la ilusión de movimiento en los *sprites*, necesitamos una forma de borrar la imagen del *sprite* que se encuentre en el *background* o fondo de pantalla antes de comenzar a dibujarlo en su nueva localización. Podríamos repintar el *background* por completo y dibujar el *sprite*, pero se perdería mucho rendimiento debido a la gran cantidad de información a procesar por el tamaño de las imágenes.

Existe un truco que consiste en copiar el trozo que va a ocupar el *sprite* antes de que se visualice, y después de ello, volver a poner ese trozo copiado procediendo a copiar el siguiente en la nueva localización del *sprite*. Este método es conocido como "*patching*". Este proceso trabaja rápido porque no emplea tanto tiempo de proceso en el volcado de gráficos al utilizar porciones más pequeñas de la imagen en vez del *background* entero.

Figura 3.- Representación de la técnica del Clipping.



A continuación se detallan los 5 simples pasos para poder utilizar la técnica del *patching*:

1. Asignamos el rectángulo que servirá como parche en la última posición del *sprite*.
2. Parcheamos el *background* en esa posición volcando el trozo de imagen correspondiente del *offset-screen* al *background*.
3. Actualizamos el rectángulo destino del *sprite* para reflejar su nueva localización.
4. Volcamos el *sprite* a su rectángulo actualizado en el *background* o *buffer* de vídeo.
5. Repetir todo el proceso.

Clipping

El *clipping* es una técnica utilizada para limitar la visualización de un *sprite* en la pantalla. Supongamos que hemos establecido la resolución de nuestro juego a 320x200. Si dibujamos una nave espacial que "mide" 640x480 *píxels*, obtendremos un error en DirectX puesto que el gráfico que queremos representar es mayor que la resolución de pantalla. Utilizando la técnica del *clipping*, se recortará automáticamente la parte o partes del *sprite* que no van a ser visualizadas y se imprimirá el trozo del gráfico que tiene que verse por pantalla.

En DGC se ha programado el código necesario para efectuar el *Clipping* al utilizar *sprites*, aunque algunas funciones todavía no lo soporten.

El clipping es una técnica utilizada para limitar la visualización de un sprite en la pantalla

DirectDraw posee también unos objetos determinados llamados *Clippers*. Podemos utilizar estos *Clippers* para designar ciertas áreas del *buffer* primario en modo escritura. DirectDraw efectúa operaciones de *blit* en esas áreas, protegiendo los *píxels* que se encuentren fuera del rectángulo asignado para el *Clipping*.

La ilustración muestra el funcionamiento de este estilo de *Clipping*:

DirectDraw utiliza los *Clippers* para poder representar los gráficos en modo ventana. De este modo, se encarga automáticamente de rellenar la ventana de nuestra aplicación con los datos gráficos que enviemos.

Como ya vimos en el fascículo anterior, mediante la librería de imágenes de DGC podemos guardar todos los gráficos de *sprites* creados y así manipularlos de forma sencilla en lugar de ir cargando las imágenes en diferentes *buffers* de memoria. Podemos cargar tanto *keys* individuales de *sprites* como imágenes enteras albergando todos los *keys* posibles.

En los siguientes extractos de programa (incluído de forma completa en el CD de la revista) veremos el uso de la librería de imágenes para la visualización de *sprites*, la detección de colisiones y el manejo del teclado:

```
procedure TForm1.DGCScreen1Initialize(Sender:
    TObject);
begin

  DrawFrame;

  DGCScreen1.FadePaletteOut(1);

  DGCScreen1.Flip;

  DGCScreen1.FadePaletteIn(100);

  DGCScreen1.Back.BltFast(0, 0,
     DGCScreen1.Front,
       DGCScreen1.Front.ClientRect, False);

  x := 150;
  y := 200;
  px := 100;
  yvel := 1;
  xvel := 2;
  KillDuration := 0;

  DGCScreen1.FlippingEnabled := True;
end;
```

Este primer procedimiento se encarga de inicializar diversos parámetros del programa. El procedimiento *DrawFrame* es utilizado para volcar el texto que se visualizará como *background*

en el monitor. Como ya sabemos, *FadePalleteOut* y FadePalleteIn establecen la paleta gráfica al color negro y la restauran a sus valores previos respectivamente.

Mediante el patching y clipping logramos velocidad en la representación gráfica

Bltfast se utiliza para efectuar el volcado del texto gráfico creado hacia el *buffer* secundario. Las siguientes variables son necesarias para el movimiento del "monstruo" por la pantalla, incluyendo sus coordenadas, velocidad, etc. El método *FlippingEnabled* permite el volcado entre el secundario y el primario cuando se utilice el método *Flip*.

```
procedure TForm1.FormKeyPress(Sender:
      TObject; var Key: Char);
begin

   if Key = #27 then
   begin
      DGCScreen1.FadePaletteOut(100);
      Close;
   end;
end;
```

Este procedimiento es el encargado de finalizar el programa cuando se pulse la tecla *ESC*. Si se cumple este hecho, se creará un fundido de la paleta gráfica y se cerrará el programa de forma automática.

```
procedure TForm1.DrawFrame;
begin
   with DGCScreen1.Back.Canvas do
   begin
      Brush.Style := bsClear;
      Font.Size := 24;
      Font.Color := clBlue;
      TextOut(110, 60, 'Delphi Games Creator');
      Font.Color := clTeal;
      TextOut(108, 58, 'Delphi Games Creator');
      Font.Size := 12;
      Font.Color := clYellow;
      TextOut(130, 150, 'Simple demo con detección de colisión. Usa los cursores');
      TextOut(160, 170, 'para mover la nave.
      Presiona Escape para Salir');
      Font.Color := clWhite;
      Font.Size := 8;
      TextOut(0, 467, 'Copyright 1.997
      Constantino Sánchez Ballesteros');
      Release;
end;
end;
```

En este procedimiento (*DrawFrame*) podéis observar la utilización de la clase *Canvas* en DGC para poner texto en nuestro *buffer* secundario. Para ello, es necesario hacer la llamada de la siguiente forma:

```
DGCScreen1.Back.Canvas
```

Seguidamente, no tenemos más que asignar *Textout* al método anterior para asignar el texto correspondiente, incluyendo el tipo de fuente utilizada, color, etc.

El siguiente procedimiento (*DGCscreen1Flip*) lo iremos desglosando para una mejor comprensión de todos los aspectos que incluye:

```
procedure TForm1.DGCScreen1Flip(Sender:
      TObject);
var
  ShipHit: Boolean;
begin

   DGCScreen1.Back.EraseRect(Rect(0, 200, 640,
      468), 0);

   DGCScreen1.Back.Draw(x, y,
      DGCScreen1.Images[0], True);
```

La primera instrucción que nos encontramos utiliza el método *EraseRect*. Su fin es borrar el espacio de pantalla por el cual se moverán los *sprites* creados. Si esto no se hiciera así, todos los *sprites* harían un efecto en pantalla de estela, puesto que dejarían huella a su paso cuando efectuaran desplazamientos en la pantalla. Puesto que este es un ejemplo sencillo he utilizado *EraseRect*, aunque no es nada recomendable para la creación de videojuegos profesionales puesto que de fondo siempre suele haber un gráfico o un *scroll*. Cuando vayamos avanzando en el curso veremos como crear fondos gráficos que no se borran al paso de los *sprites* por la pantalla.

- **EraseRect** : Borra un rectángulo de un *buffer* con el color especificado. Para asignar las coordenadas se utiliza una variable de tipo *Rect* y para el color del borrado un número de 0 a 255 que se corresponderá al asignado en la paleta gráfica que se esté utilizando.

EraseRect tiene como fin borrar el espacio de pantalla por donde se moverán los sprites

El método *Draw* es el encargado de dibujar el *sprite* seleccionado. En el ejemplo, se utiliza un *sprite* cargado en la librería de imágenes asignada al componente *DGCScreen1*. Para ello, se asigna el método *Images[x]* a *DGCScreen1*, donde X selecciona el *key* correspondiente. En este caso, X vale 1, y por ello selecciona el gráfico correspondiente al "monstruo".

- **Draw** : Este método dibujará el *buffer* gráfico seleccionado en las coordenadas X, Y asignadas. Para lograr el efecto de transparencia anteriormente comentado, debemos asignar el valor *true* al último parámetro del método. Os recuerdo que el *sprite* sólo será transparente si *TransparentColor* ha sido asignado al *buffer* que contiene la imagen. *Draw* efectúa el *clipping* de forma automática, por lo tanto, no necesitamos crear una rutina especial para implementarlo.

```
Inc(x, xvel);
  if (x < 0) or (x > 500) then
```

```
    xvel := -xvel;


  Inc(y, yvel);


  Inc(yvel);
  if yvel > 18 then yvel := 18;
  if yvel < -18 then yvel := -18;

  if y > 380 then
  begin
    y := 380;
    yvel := -yvel;
  end;
```

En este extracto del procedimiento se efectúa el seguimiento a las coordenadas X e Y de los *sprites*. Con el método *INC* (estándar de Delphi) se incrementan las variables para asignar las nuevas coordenadas. Xvel e Yvel controlan el rebote y la velocidad a la que se moverán los *sprites* en un momento dado.

La variable killduration se encarga de establecer el tiempo que nuestra nave experimenta la colisión

El siguiente paso a dar es testear la colisión entre el "monstruo" y nuestra nave espacial. Para ello, utilizamos una nueva instrucción llamada **CollisionTest:**

```
  if DGCScreen1.Images[0].CollisionTest(x, y,
     DGCScreen1.Images[1],
   px, 450, True) then
   Killduration := 10;

  if KillDuration > 0 then
  begin
   DGCScreen1.Back.Draw(px, 430,
     DGCScreen1.Images[2], True);
   Dec(KillDuration);
  end
  else
   DGCScreen1.Back.Draw(px, 430,
     DGCScreen1.Images[1], True);
```

La variable *killduration* se encarga de establecer el tiempo que nuestra nave experimenta la colisión. Si la colisión se establece entre los dos *sprites*, *killduration* valdrá 10 y posibilitará que aparezca un nuevo *sprite* de la nave con un golpe en su frontal. Cuando *killduration* llegue otra vez al valor de 0 se restaurará la antigua apariencia de la nave. Este hecho permite la posibilidad de animación en la colisión.

Cuando se selecciona el valor 1 en la librería de imágenes se activa la nave en su estado normal, y cuando se selecciona el valor 2 aparece la nave como si la hubiesen golpeado.

Cuando se gestionan colisiones, debemos prestar mucha atención a las coordenadas de los sprites implicados

La sintaxis utilizada para el método *collisiontest* es como sigue:

- **CollisionTest (X, Y: Integer; SourceSurface: TDGCSurface; SX, SY: Integer; PixelTest: Boolean): Boolean** : Como se puede apreciar al final de los parámetros del método, se trata de una función *booleana*. *CollisionTest* chequeará si la imagen que tomamos como referencia al pasar el método se superpone en sus coordenadas X y con la imagen correspondiente al *buffer* *SourceSurface* en sus coordenadas SX, SY. Si existe colisión, la función retornará el valor *True*.

Si el parámetro *PixelTest* es asignado como *false*, una colisión es retornada si cualquier región de las dos imágenes se superponen en sus coordenadas (en este caso, la función *IntersectRect* de Windows es llamada). Si *PixelTest* es asignado como *true*, la región superpuesta de las imágenes es chequeada con un color no transparente en la misma posición de cada imagen. El valor establecido por defecto en *TransparentColor* es 0 salvo que se indique otro distinto.

Finalizando la lectura del presente procedimiento, se establecen las condiciones que permiten mover la nave de un lado a otro de la pantalla. Se han definido 2 límites (2 y 605) para su coordenada X mediante la variable PX, que permiten el movimiento de la nave sin salirse de los límites visibles de la pantalla.

```
  if (DGCScreen1.KeyDown(VK_LEFT)) and (px
    > 2) then
   Dec(px, 2)
  else
    if (DGCScreen1.KeyDown(VK_RIGHT)) and
    (px < 605) then
     Inc(px, 2);
end;
```

El último procedimiento, *DGCScreen1Paint*, se encarga de redibujar la pantalla cuando se efectúa un cambio de aplicación en Windows. Este procedimiento ya ha sido comentado en el número anterior y por eso no volveré a entrar en detalles.

```
procedure TForm1.DGCScreen1Paint(Sender:
     TObject);
begin
  DrawFrame;
  DGCScreen1.Flip;
  DrawFrame;
end;

end.
```

Stretching

Una de las posibilidades que nos permite DirectDraw es la funcionalidad del *stretching*. Éste consiste en agrandar o encoger un gráfico al tamaño deseado. Para ello, se toma el rectángulo que alberga la imagen original, y a continuación se vuel-

ca dicho rectángulo pero con dimensiones diferentes. Puesto que la imagen se adaptará al último rectángulo creado, se podrán lograr efectos de escalado en diversos tamaños.

Los afortunados poseedores de tarjetas gráficas aceleradoras de última generación seguramente tengan soporte de *stretching* mediante *hardware*, sumándole el consiguiente aumento de velocidad en la representación, aunque el resto de usuarios con tarjetas menos potentes también podrán utilizar esta técnica debido a que DirectDraw la emula mediante *software*. Siempre será preferible que las funciones sean soportadas por *hardware* para ganar en rendimiento y velocidad de ejecución, además de liberar líneas de código al programador.

Figura 4.- Ejemplo de stretching.

Imagen original

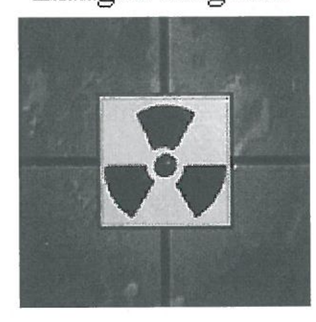
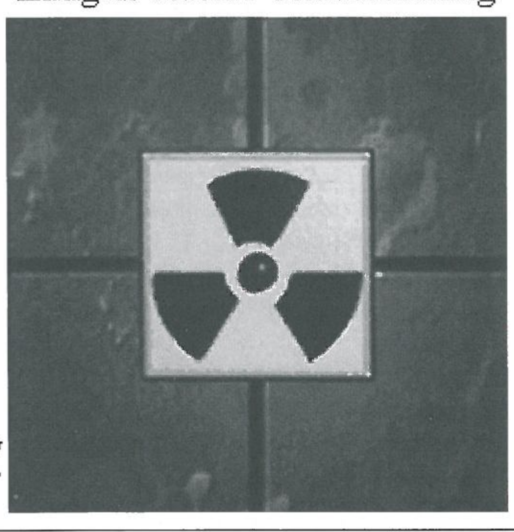


Imagen destino con stretching



DETALLE DEL EFECTO "STRETCHING"

Si las funciones son soportadas por hardware se obtiene un mayor rendimiento y velocidad

- **Factores mínimos y máximos del stretching**
  Debido a las limitaciones *hardware* antes comentadas, algunos dispositivos restringen la amplitud que puede tener el rectángulo destino para poder establecer la comparación con el rectángulo original (recordemos que los rectángulos son igualmente utilizados en la representación de *sprites*, aunque parezca que tienen sus propios perfiles). DirectDraw comunica estas limitaciones como factores de *stretch*. Un factor de *stretch* es el ratio entre el ancho del rectángulo original y el de destino. Si el *driver* instalado envía información sobre factores de *stretch*, éstos se detallarán como múltiplos de 1000; por consiguiente, un valor de 1300 significaría 1.3 (y 750 debería ser 0.75).

Los dispositivos que no impongan límites en el *stretching*, frecuentemente informan de los factores máximo y mínimo de *stretch* mediante el valor 0. El valor mínimo del factor de *stretch* nos dice cuánto más ancho o estrecho necesita ser el rectángulo destino comparado con el original. Si el mínimo factor de *stretch* es mayor que 1000, debemos incrementar el ancho del rectángulo destino por ese ratio (1000).

Por ejemplo, si el *driver* nos reporta un factor de 1300, debemos asegurarnos que el rectángulo destino es al menos 1.3 veces mayor o menor que el rectángulo original.

Después del *stretching*, el rectángulo destino debe acatar cualquier restricción de alineamiento que el dispositivo pueda requerir. De todas formas, es una buena idea efectuar el *stretching* del rectángulo destino antes de que sea alineado al tamaño correcto definitivo. El *hardware* no requiere que se ajuste el alto de los rectángulos destino. Entonces, de esta manera podemos incrementar su altura para respetar el ratio del aspecto sin efectos negativos.

La técnica del stretching es conveniente utilizarla con soporte hardware debido al consumo de tiempo que emplea el 'Micro'

En lo concerniente a la programación del *stretching*, podemos obtener el efecto en DGC vía *StretchDraw*:

- **StretchDraw(TheRect: TRect; SourceSurface: TDGCSurface: Transparent: Boolean)** : Dibujará el *buffer SourceSurface* en el rectángulo especificado por la variable *therect* efectuando el *stretching* de la imagen de forma acorde. Si *Transparent* es asignada como *True*, el *buffer* es visualizado de forma transparente. La imagen origen sólo será visualizada de forma transparente si *TransparentColor* a sido asignado al *buffer*. Este método no permite *clipping*, y por lo tanto, no se debe asignar la imagen resultante fuera de los límites de la pantalla. En

el caso que la tarjeta gráfica sí lo soporte, este impedimento será solucionado mediante *hardware* de forma automática. Este hecho nos demuestra la versatilidad de los nuevos dispositivos permitiendo más funciones que el simple soporte *software*.

La variable *TheRect* es de tipo rectángulo y por lo tanto debe incorporar cuatro coordenadas, X - X1 - Y - Y1. El siguiente ejemplo muestra el uso ficticio de este método:

```
StretchDraw(Rect (0,0,200,200), mySurface, true)
```

En este ejemplo estableceremos el nuevo tamaño de la imagen contenida en el *buffer mysurface* a 200 *píxels* de ancho y 200 *píxels* de alto. Con el valor *true* al final del método, aplicamos transparencia a la imagen.

DirectX & COM

Tal y como venimos haciendo de forma continuada, además de aprender a programar las librerías DirectX, vamos introduciéndonos poco a poco en las características, estructuras y funcionamiento interno del API. Esto nos ayudará a comprender muchos conceptos que pueden resultar confusos en un principio pero que serán bienvenidos a la hora de programar.

La mayoría de los API´s incluidos en DirectX están compuestos de objetos e interfaces basados en COM. El COM es una fundación creada para un sistema basado en objetos que convergen en la reutilización de *interfaces*. Éste es el modelo utilizado en la programación COM. Podemos definirlo como una especificación con la que cualquier número de *interfaces* pueden ser construidos.

Muchos API´s de DirectX son creados como instancias de objetos COM. Podemos considerar un objeto como una caja negra que representa el *hardware* y requiere comunicación con las aplicaciones a través de un *interface*. Los comandos enviados y recibidos por el objeto a través del *interface* COM son llamados métodos. Por ejemplo, el método *GetDisplayMode* es enviado a través del *interface* DirectDraw2 para tomar el valor de la actual resolución de la pantalla mediante el objeto DirectDraw.

La variable TheRect debe incorporar cuatro coordenadas

Los objetos pueden vincularse unos a otros en tiempo de ejecución, y pueden usar la implementación de *interfaces* provistos por otros tantos objetos diferentes. Si sabemos que un objeto es del tipo COM y qué *interfaces* soporta, nuestro programa (u otro objeto) puede determinar qué servicios ofrece el primer objeto. Uno de los métodos heredados en todos los objetos COM, en concreto el método *QueryInterface*, nos permite determinar qué *interfaces* soporta un objeto y crea los correspondientes punteros.

Interfaces COM en DirectX

Los *interfaces* de DirectX han sido creados a un nivel muy básico de la jerarquía empleada en la programación COM. Cada *interface* corresponde a un objeto que representa un dispositivo, como pueden ser DirectDraw, DirectSound y DirectPlay, derivando directamente del *interface* COM *IUnknown*. La creación de estos objetos básicos es manipulada por funciones especializadas de la correspondiente DLL.

Típicamente, el modelo de objeto empleado por DirectX permite un objeto principal por cada dispositivo. El soporte para otros objetos distintos es derivado del principal.

Por ejemplo, el objeto DirectDraw representa la tarjeta gráfica. Con éste, podemos crear objetos del tipo DirectDrawSurface que representan la memoria de vídeo y objetos DirectDrawPalette que representan las paletas gráficas que utilicemos en los gráficos. De forma similar, el objeto DirectSound representa la tarjeta de sonido y crea objetos del tipo DirectSoundBuffer representando los datos de sonido que posteriormente se enviarán a la tarjeta.

Además de la habilidad para generar objetos subordinados, el objeto principal determina las posibilidades del dispositivo *hardware* que representa, como puede ser la resolución de pantalla y el número de colores, o si la tarjeta de sonido permite síntesis de tabla de ondas.

Utilización de múltiples monitores

El futuro Windows 98 y Windows NT 5.0 soportarán múltiples tarjetas gráficas y monitores en un mismo PC. La arquitectura del múltiple monitor (técnicamente llamada "*Multimon*") permite al sistema operativo utilizar el área de visualización de 2 o más tarjetas gráficas y monitores para crear un escritorio común. Por ejemplo, en un sistema *MultiMon* de 2 monitores, el usuario podría visualizar diferentes aplicaciones o videojuegos en cada uno. DirectDraw ofrece soporte para esta tecnología, pero hay todavía algunos flecos que depurar dependiendo del nivel de cooperación que utilicen nuestros programas de DirectX.

Una aplicación creada con DirectDraw debería enumerar los dispositivos, elegir uno (o permitir que lo eligiera el usuario), y entonces crear un objeto DirectDraw para cada dispositivo utilizando el identificador global del *hardware* seleccionado (GUID). Con esta técnica, nos aseguraremos el mejor rendi-

miento en un solo monitor, sistemas *Multimon*, y sus niveles de cooperación.

La tarjeta gráfica activa es referida como *"default device"* (dispositivo por defecto) o *"null device"* (dispositivo nulo). Este segundo nombre viene del hecho de que el dispositivo gráfico actual es enumerado como *NULL* por el GUID. Muchos programas crean objetos DirectDraw para el *"null device"*, asumiendo que el dispositivo será acelerado por *hardware*. Debemos tener en cuenta que en sistemas *Multimon*, el *"null device"* no siempre es acelerado por el *hardware* instalado en nuestro PC; depende de qué nivel cooperativo se haya asignado.

En un nivel cooperativo *"ful screen* modo exclusivo", el *"null device"* es acelerado por *hardware*. Esto significa que este nivel cooperativo correrá más rápido en sistemas *Multimon* que en cualquier otro, pero no será capaz de utilizar el soporte de construcción para operaciones gráficas de un monitor a otro. Los programas creados en *"Full screen* modo exclusivo" que necesiten utilizar múltiples dispositivos, pueden crear un objeto DirectDraw por cada dispositivo que quieran utilizar.

Mediante la revolucionaria técnica del Multimon, podemos visualizar al mismo tiempo múltiples aplicaciones en diversos monitores

Cuando el nivel cooperativo es asignado como normal, el *"null device"* no tiene aceleración *hardware*; este dispositivo es, efectivamente, un dispositivo lógico pero emulado, que combina los recursos de 2 dispositivos físicos. Por otro lado, cuando el nivel cooperativo es asignado como normal, el *"null device"* es capaz de utilizar el soporte de construcción para operaciones gráficas de un monitor a otro. Como resultado, las coordenadas negativas para operaciones de *blit* son válidas cuando la localización lógica del monitor secundario esté a la izquierda del monitor primario.

Si nuestra aplicación requiere aceleración por *hardware* cuando el nivel cooperativo es asignado como normal, debemos crear un objeto DirectDraw utilizando un específico dispositivo GUID. Como regla en cualquier sistema, deberíamos asignar el nivel cooperativo inmediatamente después de crear un objeto DirectDraw, antes de requerir las características del objeto, o utilizar otros *interfaces*. Si necesitamos cambiar el modo de vídeo de *full-screen* a normal, es mejor crear un nuevo objeto DirectDraw.

Sistemas Distribuidos (III): El Run Time System

Jorge F. Delgado Mendoza (ernar@convex.es)

Después de los dos artículos teóricos anteriores, en los que hemos aprendido conceptos básicos que nos van a permitir la construcción de un Sistema Distribuido, ha llegado el momento de ponernos manos a la obra. Para abrir boca, vamos a comenzar con el componente más importante de todos: el *Run Time System.*

Para comenzar la implementación de un Sistema Distribuido, hemos dedicado dos artículos a aprender una serie de conceptos básicos sobre la interconexión cooperativa de un grupo de ordenadores.

En el primer artículo, publicado en el Número 40 de la revista, establecimos el marco en el que nos íbamos a mover. Para ello enumeramos y describimos los diferentes puntos que han de ser tenidos en cuenta a la hora de diseñar un Sistema Distribuido:

- Estudiamos diferentes maneras de organizar los datos de tal manera que todas las máquinas del entorno accedan a los mismos datos.
- Describimos, en detalle, el concepto de *Modelo de Coherencia* y analizamos todas las posibles opciones que teníamos para mantener coherente nuestro sistema de Memoria Distribuida.
- Analizamos varias estrategias de *Replicación de Datos*, con el fin de optimizar al máximo el rendimiento del sistema final.
- El análisis anterior nos llevó a plantearnos la unidad mínima de información que viajaría a través de la red, al ver cómo la elección de un tamaño adecuado evitaba situaciones que comprometían en gran medida el comportamiento del sistema.

En el segundo de los artículos - Solo Programadores N°41 - nos planteamos la construcción de un Sistema Distribuido. Para ello realizamos un análisis de requisitos y establecimos una serie de Condiciones que tendría que cumplir el entorno que íbamos a construir:

- Alto índice de Conectividad.
- Elevado número de Usuarios.
- Sistema Heterogéneo.
- Diseño Modular.
- Facilidad de Uso: Modelo de *Máquina Virtual Única*.

Para satisfacer todas estas condiciones, y algunas más que añadíamos en el artículo, establecíamos la arquitectura de nuestro Sistema Distribuido, decidiendo que modelo íbamos a utilizar en cada uno de los aspectos relevantes del sistema.

En cuanto a las *comunicaciones* entre las diferentes máquinas y los distintos elementos del entorno, decidimos utilizar *Remote Procedure Calls* por motivos de sencillez de codificación y de portabilidad.

El *Modelo de Memoria* a utilizar sería el de *Variables Compartidas*, el cual nos permitía una gran flexibilidad en el manejo de los datos, sin por ello sacrificar la sencillez a la hora de utilizar nuestro Sistema Distribuido.

Finalmente, decidimos incorporar un *Modelo de Coherencia* basado en lo que definimos como *Consistencia Secuencial*. De esta manera, aunque nuestro sistema

no sea tan rápido como podría llegar a ser, facilitamos enormemente el trabajo de la persona que vaya a utilizar el entorno distribuido que vamos a crear.

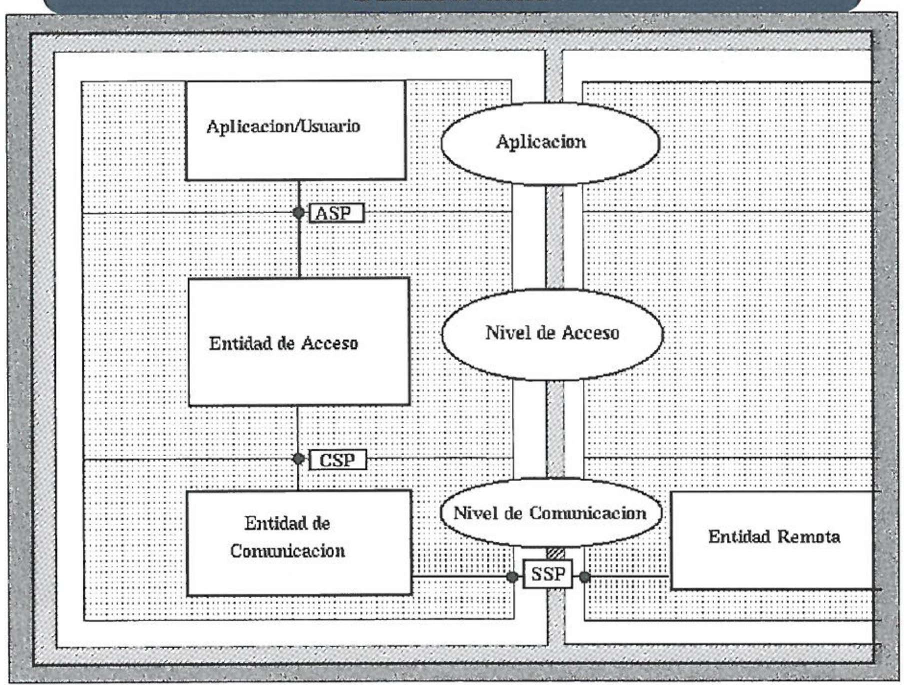
Este Entorno Distribuido va a constar de dos componentes: el RTS y una librería de comunicaciones.

- El **RTS** *(Run Time System)* va a ser el encargado de mantener la coherencia entre las diferentes copias de la información presentes en cada una de las máquinas del sistema.
- La **Librería de Comunicaciones.** Es la parte de Sistema Distribuido que va a ver el usuario. Va a incluir los puntos de acceso al sistema de Memoria Compartida, bastando incluirla en cualquier programa para poder disfrutar de los servicios del Sistema Distribuido.

En el presente artículo vamos a construir la parte más importante del sistema: el RTS. Su misión exige que se esté ejecutando permanentemente en el ordenador, - al menos mientras deseemos utilizar el entorno distribuido, - por lo que lo vamos a construir como si de un demonio más del sistema se tratase. Las peticiones que realicen las aplicaciones se efectuarán mediante llamadas *cliente/servidor*, en las cuales el RTS será el servidor y los programas que utilicen el Sistema Distribuido, los clientes.

Funcionalmente, en RTS o *servidor* del sistema podemos identificarlo con la *Entidad de Comunicaciones* de la cual hablábamos en el artículo anterior, ofreciendo a la Librería los servicios de la interfase CSP y a los demás servidores del sistema los de la interfaz SSP.

Figura 1. Descomposición en capas del Sistema Distribuido. El RTS del sistema, el cual se ha de estar ejecutando siempre, se puede identificar con la Entidad de Comunicaciones, mientras que la Librería de Comunicaciones se corresponde con la Entidad de Acceso.



Requisitos

- **Conocimientos:** todo el sistema va estar programado en C, por lo que es necesaria cierta experiencia programando en este lenguaje. Es también muy importante haber entendido los conceptos básicos que se han expuesto en los dos artículos anteriores, publicados en los números 40 y 41, ya que a lo largo del proceso de construcción del código vamos a realizar numerosas referencias a estos conceptos.

Finalmente, aunque en este artículo vamos se va a explicar por encima el paradigma de *Remote Procedure Call*, un análisis detallado del mismo se escapa de los objetivos de este artículo y, - efectivamente, - es necesario saber qué es una RPC, para qué sirve y cómo se utiliza.

- **Software:** necesitaremos las siguientes herramientas instaladas en todos los ordenadores del sistema:
- El compilador GCC, versión 2.7.0 ó superior.
- La herramienta MAKE.
- El *software* de RPC's.
- El *software* de TCP/IP.

Además, es necesario que el demonio de RPC, llamado *portmapper*, se encuentre activo siempre que queramos utilizar el sistema. Para activarlo, podemos hacerlo a mano como *root* cada vez que lo necesitemos, aunque lo ideal es añadir al fichero de configuración de red las siguientes líneas:

```
# Start the SUN RPC Portmapper.
if [ -f ${NET}/rpc.portmap ]; then
  echo -n " portmap"
  ${NET}/rpc.portmap
fi
```

El fichero de configuración al que nos referimos es el que se ejecuta al iniciar el sistema y el *path* completo hasta él es:

```
/etc/rc.d/rc.inet2
```

Una vez realizada esta operación, cada vez que reiniciemos el ordenador, dispondremos de los servicios asociados a las RPC's.

- **Documentación:** prácticamente todos los conceptos que se van a utilizar se han explicado en los dos

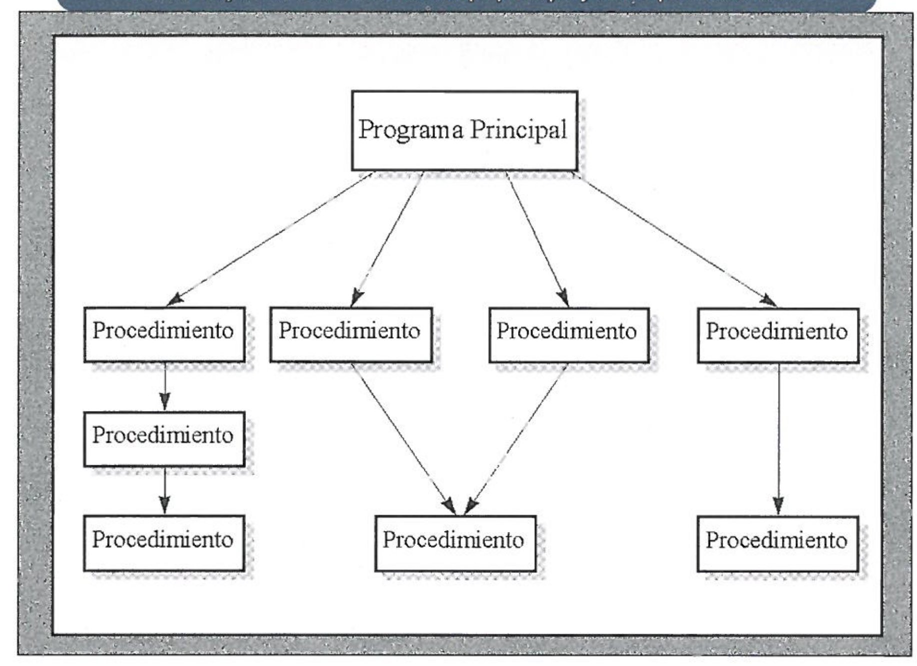


Figura 2. Esta figura describe la estructura intérna de un programa, en el cuál se pueden identificar un cuerpo principal y varios procedimientos.

artículos anteriores, por lo que tener a mano una copia de ambos artículos será de gran ayuda. Sin embargo, todo aquel que quiera profundizar en el campo de los Sistemas Distribuidos, puede utilizar las referencias que aparecen en el apartado dedicado a bibliografía.

En cuanto a información sobre las Llamadas a Procedimiento Remoto (RPC's), el libro de cabecera para todo aquel que se dedique a trabajar con ellas es el titulado:

"Internetworking With TCP/IP Volume III:
Client/Server Programming and
Applications"
Douglas E. Comer, David L. Stevens.
Editorial Prentice-Hall International.

Aunque desgraciadamente está en Inglés su redacción es muy amena, ilustrando de una manera práctica todos los pasos necesarios para la programación de aplicaciones que necesiten comunicarse mediante protocolos de la familia TCP/IP.

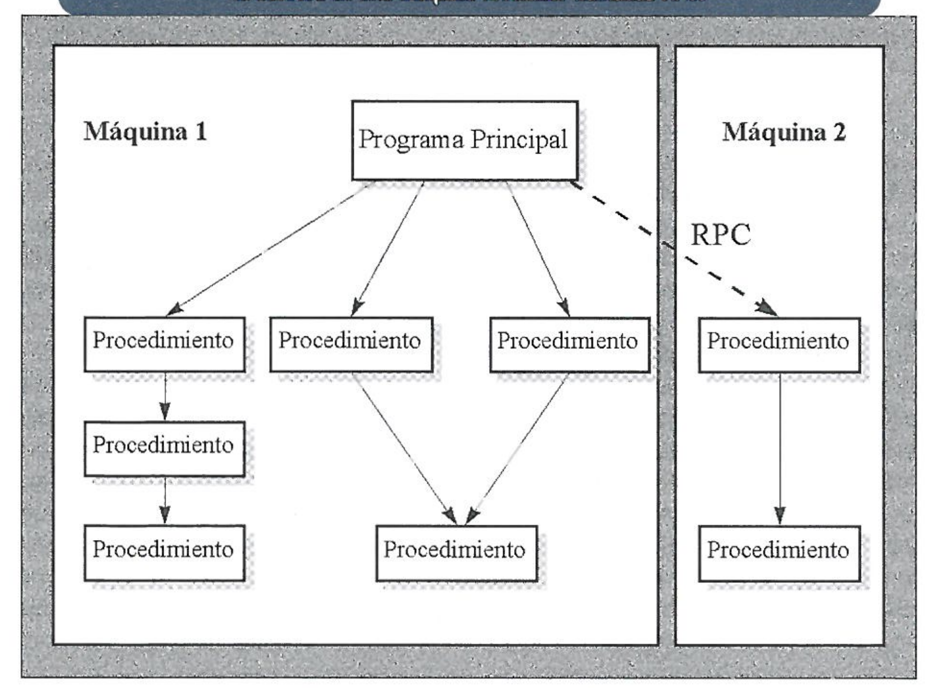


Figura 3. Estructura interna de un programa que traspasa la barrera de una máquina mediante llamadas RPC.

Llamadas a Procedimiento Remoto (RPC's)

La principal tentación del programador a la hora de construir aplicaciones que van a operar en varias máquinas simultáneamente, es la de construir un protocolo de comunicaciones eficiente y rápido, aunque lo suficientemente flexible como para que sea útil.

Aunque a primera vista esta aproximación parece válida, una atención excesiva a resolver el problema de las comunicaciones convierte al protocolo en la pieza central del sistema. Este hecho suele convertir a la aplicación en una pesadilla a la hora de buscar errores, ya que las interacciones entre los diferentes componentes de las diferentes máquinas convierten el proceso de depuración en una carrera de obstáculos, donde las vallas son los saltos entre ordenador y ordenador.

Como desventaja añadida, tenemos que un excesivo interés en conseguir un protocolo de gran rendimiento puede condicionar la implementación de la aplicación en sí, llegándose a soluciones poco óptimas al problema que queremos resolver. El medio, la distribución de información y proceso, se convierte en

más importante que el fin (las soluciones que brinda la aplicación).

Para evitar estos inconvenientes, el paradigma RPC propone un método distinto de realizar aplicaciones en el cuál prima la resolución problema. Según la metodología RPC, lo primero que hemos de hacer es la aplicación objetivo como si fuéramos a ejecutarla en una única máquina. De esta manera se permite al programador concentrarse en la arquitectura del programa en sí, antes de dividirlo en piezas que se ejecuten en diferentes ordenadores.

Una vez tenemos una aplicación centralizada, - cuya estructura será similar a la representada en la Figura 2, - que cumple los objetivos de diseño es el momento de empezar a pensar el la distribución de los componentes.

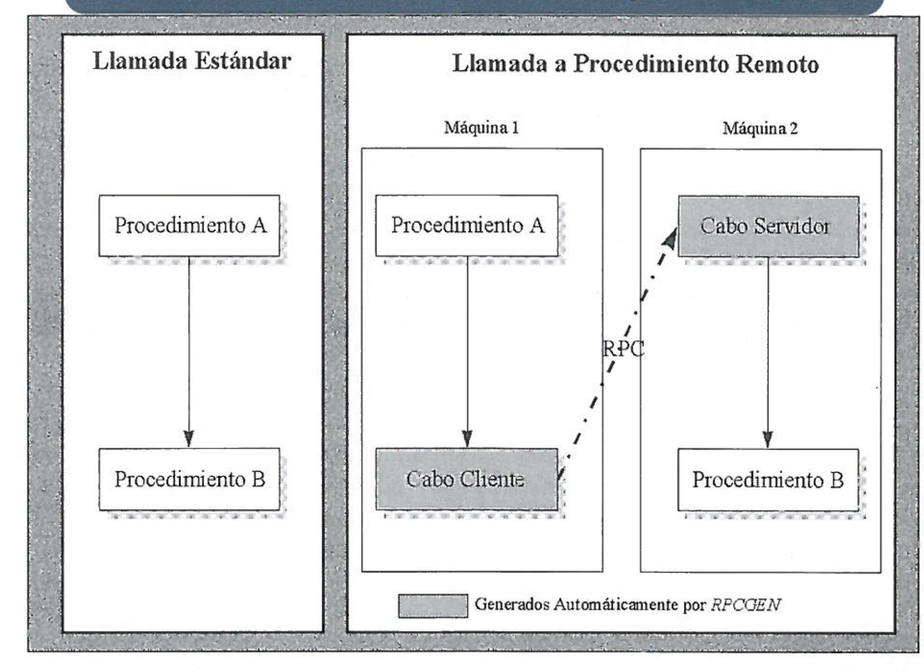
El sistema de RPC's nos proporciona un mecanismo para salvar la barrera que existe entre dos máquinas conectadas mediante una red TCP/IP. Utilizando este método una aplicación podrá llamar a procedimientos que ejecuten en otra máquina distinta, mediante la utilización de un protocolo transparente y estandarizado.

Una nueva versión del programa de la Figura 2, distribuida mediante llamadas RPC, se puede ver en la Figura 3.

La utilización de los protocolos RPC es prácticamente transparente al programador, ya que casi todo el trabajo se realiza de manera automática gracias a la herramienta *rpcgen* (*RPC GENerator* o Generador de Llamadas a Procedimiento Remoto).

Mediante esta herramienta construimos dos cabos, uno en el lado del cliente y otro en el lado del servidor, los cuales nos permiten realizar las llamadas localmente, como si de procedimientos estándar se tratase. Estos cabos se van a encargar de todas las tareas asociadas con el intercambio de información entre las máquinas. Para ello, lo único que tendremos que realizar es una especificación, tanto de los datos a pasar como de los procedimientos a utilizar.

Figura 4. Modelos estándar y remoto de llamada a procedimiento. Como los cabos utilizan el mismo interfaz que la llamada original, no es necesario cambiar código en ninguno de los dos procedimientos originales.



La única limitación de la RPC es que todos los parámetros que utilicemos en las llamadas han de ser pasados por valor, y no por referencia, ya que ambos procedimientos ejecutan en espacios de memoria disjuntos.

Primer Paso: Generar la Especificación

De todo lo que hemos aprendido sobre RPC's podemos deducir que el primer paso consiste en generar la especificación de la comunicación entre procesos. Una vez realizada esta tarea será el momento de utilizar las herramientas necesarias para la generación automática de los cabos.

- **Diseño del Interfaz**

Para evitar definir un cúmulo de funciones diferentes con distintos argumentos, hemos optado por utilizar una única función para el paso de los mensajes entre los ordenadores.

Para hacer aun más claro el código, la función de contacto - **Contact** - va a tener un único argumento: la estructura *Frame*. De esta manera centralizamos todos los argumentos en un único componente, lo cual redundará en una mayor claridad estructural de nuestro código.

Los componentes de esta estructura son cuatro:

- **Frame.function**: Determina el tipo de operación a realizar.
- **Frame.coordinate**: Indica la posición de la memoria compartida sobre la que realizar la operación.
- **Frame.value**: Valor de la posición de memoria accedida.
- **Frame.auxiliar**: Valor auxiliar, utilizado para mantenimiento interno.

Dado que el fin de nuestro Sistema Distribuido es didáctico, nos vamos a conformar con tener una memoria distribuida de 100 *bytes*. Estos valores van a estar almacenados en una cadena denominada *Array* la cual contendrá, en cada una de

números atrasados

SÓLO PROGRAMADORES

completa ya tu colección



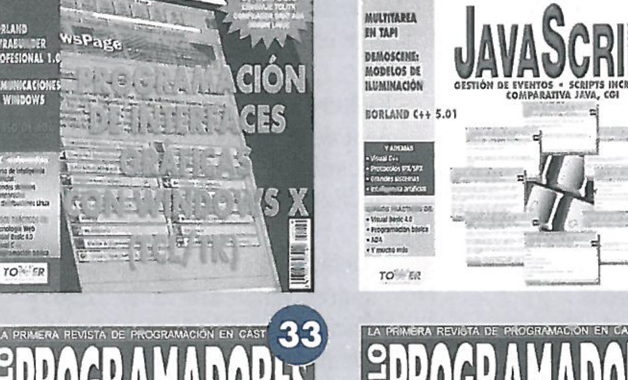




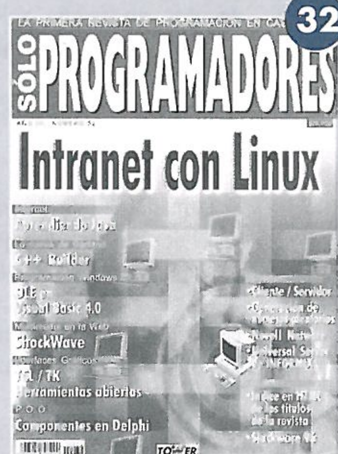




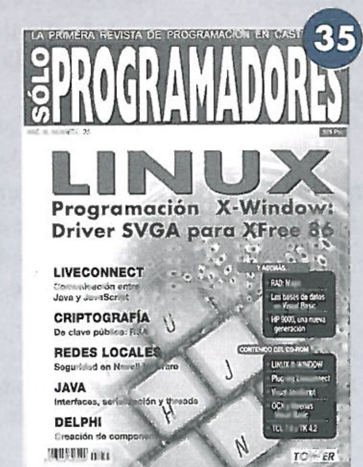


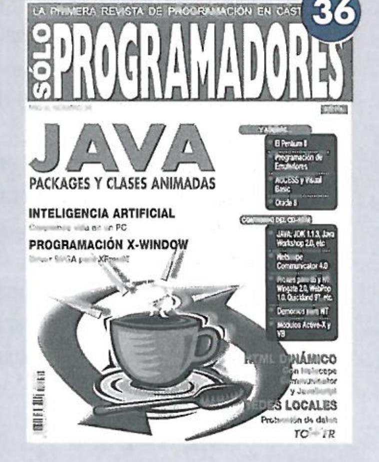












sus posiciones, los diferentes valores del espacio de Variables Distribuidas.

Existe un tercer componente de gran importancia: el cerrojo. Mediante él vamos a garantizar acceso exclusivo a la memoria compartida, garantizando la *coherencia secuencial*. El estado del cerrojo se conserva en la variable *Locks*. Para evitar que se produzcan situaciones de bloqueo - *deadlocks* - al activar los cerrojos, se elige una máquina del sistema como líder del mismo - aunque todas las máquinas tienen la posibilidad de ser líderes - y por tanto como encargada de gestionar el comportamiento de los cerrojos.

Cuando una máquina desea actualizar el valor de una variable, el procedimiento que sigue es el siguiente:

- Contacta con el líder. Si éste se encuentra bloqueado, se aborta la actualización. En caso contrario se activa el cerrojo.
- Una vez se ha colocado el cerrojo con éxito, la máquina que desea escribir un nuevo valor envía el mismo a todas las demás máquinas del sistema.
- Al finalizar esta tarea, desbloqueamos el servidor.

Si la máquina que desea escribir es el propio líder, bastará con que él mismo realice la actualización y el bloqueo.

Existen cinco tipos de mensajes, en función de los valores de *Frame.funcion*:

- **Lectura:** es el más sencillo de todos, ya que al optar en el diseño por una replicación total del espacio de Variables Compartidas, basta con realizar una lectura local para obtener el valor actualizado de la variable.
- **Escritura:** activa el conjunto de funciones necesario para actualizar el valor de una variable en todos los servidores.
- **Cerrar:** permite a una máquina contactar con el líder para activar el cerrojo y así poder realizar actualizaciones.
- **Desbloquear:** es la función especular de la anterior y permite liberar al líder, permitiendo a otras máquinas el acceso a la memoria compartida.
- **Escritura Local:** es una función interna ofrecida a los distintos servidores para actualizar los datos de las demás máquinas.

Las funciones de *Lectura* y *Escritura* se ofrecen al usuario del entorno distribuido; las otras tres se utilizan para sincronización interna y no deben ser utilizadas por las aplicaciones.

- **Utilizando RPCgen**

Una vez sabemos los datos que van a circular por la red, ha llegado el momento de construir los cabos de los procedimientos, tanto cliente como servidor. Para ello vamos a seguir los siguientes pasos:

Escribimos nuestra especificación, donde deberemos incluir los tipos de datos que van a viajar por la red, así como las llamadas que van a traspasar la barrera de la máquina. (Las peticiones de los usuarios se van a tratar también como RPC's). Llamaremos a nuestra especificación *proto5.x*.

Ejecutamos el programa *rpcgen* de la siguiente manera:

```
rpcgen -a
```

Este comando nos va a generar los siguientes ficheros:

- **proto5.h** Conjunto de definiciones necesarias para las llamadas.
- **proto5_client.c** Ejemplo de programa que utiliza RPC's para comunicarse.
- **proto5_clnt.c**Cabo de la RPC perteneciente al lado del cliente.
- **proto5_server.c** Cabo de la RPC perteneciente al lado del servidor.
- **proto5_svc.c** Servidor de la RPC.
- **proto5_xdr.c** Fichero de conversión de datos.

A la hora de construir el servidor, vamos a emplear los ficheros:

- *proto5.h* Cabeceras comunes
- *proto5_clnt.c* Para la parte del servidor que funciona como cliente.
- *proto5_svc.c* El servidor propiamente dicho.
- *proto5_xdr.c* Conversiones de datos.
- *proto5_server.c* Cabo del servidor, lo renombraremos *server_if.c*

Implementación del Servidor o RTS

En la implementación vamos a utilizar diferentes ficheros, agrupando las funciones similares en el mismo archivo. En la medida de lo posible evitaremos introducir modificaciones en los componentes que han sido generados automáticamente *por rpcgen*.

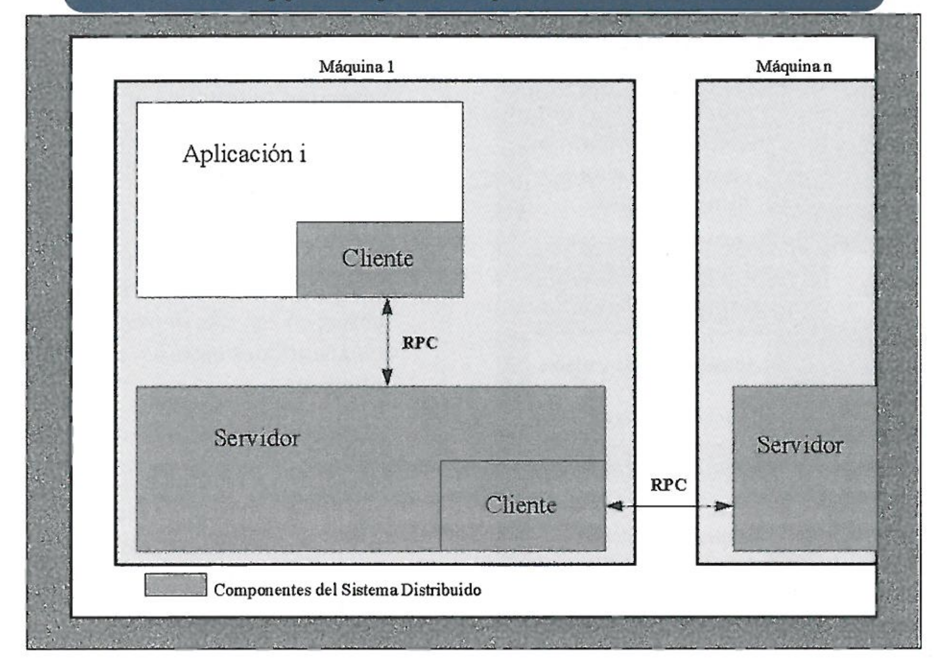
Todos los ficheros descritos en el artículo, así como los *Makefiles* necesarios para compilar el servidor, se pueden encontrar en el CD-ROM de la revista. Por este motivo, la descripción que vamos a realizar en el artículo será funcional, ya que los detalles del código se pueden ver fácilmente en los fuentes del CD.

- **Fichero common.h**
  Aquí dentro vamos a incluir todo lo que sea común al código del servidor. Entre otras, incluiremos:
  - Definiciones de las distintas constantes empleadas a lo largo y ancho del código del servidor.
  - Definiciones de las funciones que han de emplear los diferentes ficheros.

- **Fichero config.h**
  Incluimos las variables globales y definiciones de las funciones asociadas a la gestión de la tabla de máquinas que componen el sistema.

  Inicialmente, la variable *Hostlist* admite únicamente 15 máquinas en

Figura 5. En este modelo funcional de nuestro Sistema Distribuido podemos apreciar las dos partes de las que se compone el servidor. La parte de cliente es común para él y para las aplicaciones que utilicen el sistema.



el entorno. Si se desea la posibilidad de añadir más máquinas, podemos incrementar el valor de la cadena tanto como queramos.

- **Fichero server_if.c**[1]
  Contiene el cabo del lado del servidor. Su función es, simplemente, realizar la traducción de los parámetros de la llamada que vienen a través de la red en algo idéntico a lo que teníamos en la llamada desde el cliente.

  Como efecto secundario devuelve el resultado de la operación, a través de la red, hasta el cliente que realizó la llamada.

  El hecho de renombrarlo obedece a cuestiones de claridad, ya que el servidor tiene su propio fichero de código, ya que posee dos partes: la que sirve datos propiamente dicha y un cliente que le permite comunicarse con las demás máquinas.

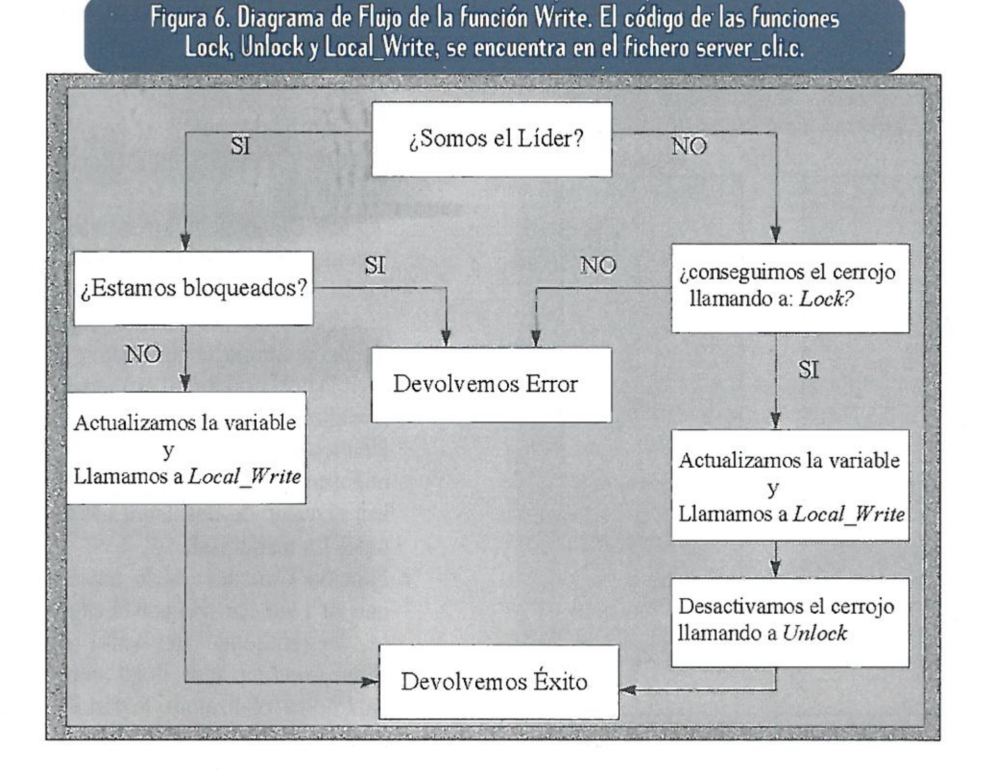
- **Fichero server.c**
  Es el código principal del servidor de nuestro Sistema Distribuido. Está compuesto por cuatro partes claramente diferenciadas:

  - Inclusión de las cabeceras que necesitamos para el código. Son:
    stdio.h Para la impresión de las trazas de depuración.
    rpc/rpc.h Necesaria en cualquier aplicación que utilice RPC's.
    string.h Tratamiento de cadenas.
    proto5.h Componentes de generación automática mediante *rpcgen.*
    common.h Componentes propios del servidor.
    config.h Componentes relativas a la configuración de las máquinas.
  - Inicialización de los componentes principales del servidor: el espacio de memoria compartida y el cerrojo del servidor. El primero se inicializa a un valor arbitrario, idéntico en todas las posiciones, mientras que el segundo se inicializa al valor de desbloqueado en todas las máquinas2.
  - Función **Contact**: es la función que va a ser llamada por el cliente. Dependiendo del valor de *frame.function* que llega como parámetro realizamos la función correspondiente. La Tabla 1 describe las diferentes opciones implementadas.
  - Función **Write**: encargada del proceso asociado a una escritura, encargándose de la gestión de cerrojos. Su diagrama de flujo es el representado en la Figura 6, que aparece en la página siguiente.

Tabla 1. Pseudocódigo de las distintas operaciones realizadas por el procedimiento remoto Contact en función del valor del parámetro frame.function.

Valor de *frame.function*	Operaciones realizadas
WRITE	Se llama a la función *Write*, pasando la posición de la variable y el nuevo valor de la misma.
READ	devolvemos en valor de la memoria compartida cuya posición corresponde al parámetro *frame.coordinate*.
LOCK	Solo va a recibir este tipo de mensajes el líder del Sistema. Si ya estaba bloqueado devolverá un código de error. En caso contrario devuelve un valor afirmativo tras bloquear el cerrojo.
UNLOCK	Desbloquea el cerrojo, devolviendo un valor de éxito.
LOCAL_WRITE	Solo puede ser utilizada por los diferentes *RTS's* que componen el sistema. Escribe en la posición *frame.coordinate* de la memoria compartida el valor pasado como parámetro.

Figura 6. Diagrama de Flujo de la Función Write. El código de las funciones Lock, Unlock y Local_Write, se encuentra en el fichero server_cli.c.



- **Fichero server_cli.c**
  Como ya hemos dicho anteriormente, cada uno de los servidores tiene una pequeña parte que funciona como cliente. Este bloque se utiliza para realizar las actualizaciones del espacio de Variables Compartidas de cada uno de los servidores presentes en el sistema.

  A todas las funciones de este fichero se accede a través del procedimiento *Write* del fichero principal del servidor: *server.c*. Estas funciones son tres:

  - *Local_Write*: encargada de actualizar el valor de una variable en los sistemas remotos.
  - *Lock:* su misión es bloquear al servidor líder para que nadie más pueda acceder a los valores de la memoria compartida.
  - *Unlock:* de manera similar a *Lock*, esta función comunica al servidor maestro que ya puede levantar el bloqueo sobre el espacio de memoria distribuida.

  La estructura de los tres procedimientos es muy similar, realizándose sus funciones en cuatro pasos diferentes:

  - El primero de ellos se limita a rellenar los campos de la estructura *Frame* que vamos a pasar como parámetro a la RPC.
  - A continuación abriremos una conexión con la máquina destinataria mediante la llamada *clnt_create*.
  - Establecida la conexión, es el momento de enviar el mensaje utilizando la función *contact_1*, creada por *rpcgen* para la parte del cliente.
  - Finalmente, cerramos la conexión mediante *clnt_destroy* y devolvemos un mensaje en función del éxito o fracaso de la operación.

  En el caso de *Lock()* y *Unlock()*, sólo vamos a abrir y cerrar un canal: el asociado al servidor líder. Sin embargo, la función *Local_Write()* abrirá y cerrará canales con todas las máquinas del sistema, ya que el nuevo valor de la variable tiene que ser distribuido por todo el entorno distribuido.

- **Fichero proto5_clnt.c**
  Aunque se genera automáticamente mediante la llamada a *rpcgen*, es necesario que lo mencionemos ya que su inclusión en el conjunto de código del servidor no es obvia. Este fichero nos proporciona un mecanismo de comunicaciones similar al de un cliente, al ofrecernos el cabo asociado al mismo.

  Al poder utilizar el cabo del cliente dentro del servidor, no tenemos que rehacer código para implementar las comunicaciones entre los diferentes servidores.

- **Fichero hosttable.c**
  Todo sistema en el que intervenga más de un ordenador debe conocer las direcciones de los demás miembros del sistema para poder comunicarse con ellos. Nuestro entorno Distribuido no es una excepción, ya que necesita información sobre las máquinas que componen el sistema para establecer los canales por los que circulan los mensajes.

  Con este fin se ha creado la variable *Hostlist*, en la cual se almacenan los nombres de las diferentes máquinas que están conectadas al Sistema Distribuido. La rutina *InitHosts* se encarga de leer la lista de servidores de un fichero de configuración llamado **hostsinit**, el cual se debe encontrar en el mismo directorio en el que ejecutamos el servidor.

  En todas las máquinas debemos utilizar el mismo fichero de configuración, - con las máquinas en el mismo orden - con una única salvedad: en cada uno de los diferentes ficheros debemos substituir el nombre de la máquina local por *"localhost"*.

- **Fichero proto5_svc.c**
  A pesar de que la política de construcción de RPC's deja bien claro que los ficheros generados automáticamente por *rpcgen* no deben ser modificados bajo ningún concepto,

existen casos en los que hemos de quebrantar esta regla.

El especial funcionamiento de nuestro sistema, en el cual hemos de realizar la inicialización de las máquinas que lo componen, nos fuerza a introducir una llamada a la función *InitHosts* en el cuerpo principal del servidor de RPC's.

Es necesario insistir: normalmente **no** debemos tocar los ficheros generados por *rpcgen* a no ser que sea absolutamente necesario. No proceder de esta manera suele llevar a errores muy difíciles de detectar a no ser que sepamos muy bien lo que estamos haciendo.

- **Más Allá**

Una vez compilado el sistema y creado el fichero con las distintas máquinas, ya estamos en condiciones de utilizar varios procesadores como si fueran uno solo de manera totalmente transparente. Ya solo nos queda construir la librería del cliente, con la cual las aplicaciones podrán hacer uso de nuestro entorno.

Aunque parezca increíble, hemos construido un sistema de estas características con apenas unos cientos de líneas de código. Sin embargo, quedan muchos puntos donde se podrían introducir mejoras, - mejoras que no se han añadido por no ser imprescindibles para la compresión del funcionamiento de un sistema de estas características.

Entre las distintas posibilidades que se nos pueden ocurrir incluimos:

- *Añadir seguridad al protocolo.* Actualmente confiamos en que los clientes no van a realizar llamadas *Lock()*, *Unlock()* y *Local_Write()*. Al protocolo de comunicaciones podríamos añadirle un servicio de autenticación gracias al cual sólo los servidores pudieran acceder a estas primitivas privilegiadas.
- *Aumentar el tamaño de la memoria compartida.* A priori basta con cambiar el tamaño del array, sin embargo como al escribir bloqueamos todo el espacio de memoria, el rendimiento cae velozmente con el aumento de tamaño. La solución consistiría en hacer bloques que se pudieran acceder independientemente.
- *Añadir soporte para Multicast.* de esta manera evitaríamos una gran mayoría de los mensajes debidos a las escrituras, ya que en lugar de enviar uno por máquina, enviaríamos una única trama que sería recibida por todos los demás servidores.
- Utilizar medios de comunicación optimizados para los accesos locales, ya que la implementación realizada utiliza RPC's para todas las conexiones independientemente de que sean locales o remotas.

Bibliografía

Todo aquel que desee conocer en más profundidad los detalles de un Sistema Distribuido puede utilizar la siguiente colección bibliográfica:

- Andrew S. Tanenbaum, *"Distributed Systems"*, Ed. Prentice-Hall International.
- Douglas E. Comer y David L. Stevens, *"Internetworking with TCP/IP Vol III"*, Ed Prentice-Hall International.
- Andrew S. Tanenbaum, *"Parallel Programming Using Shared Objects and Broadcasting"*.
- Nicholas Carriero y David Gelernter, *"How to Write Parallel Programs: A Guide to the Perplexed"*. ACM Computing Surveys, Vol 21, Nº3
- Henri E. Bal, *"A Comparative Study of Five Parallel Programming Languages"*, Proceedings of the EurOpen Spring 1991 Conference on Distributed Systems.
- C. L. Hartley y V. S. Sunderam, *"Concurrent Programming with Shared Objects in Networked Environments"*.

Conclusiones

En este tercer artículo de la serie, hemos implementado el componente más importante de nuestro Sistema Distribuido, el *Run Time System* o servidor de la Memoria Distribuida. Además, hemos aprendido a realizar Llamadas a Procedimiento Remoto o RPC's, utilizando la herramienta *rpcgen*.

Como resultado, tenemos la parte fundamental del entono distribuido de programación, el componente que se encarga de mantener coherentes los datos del sistema en todas y cada una de las máquinas.

En el próximo artículo completaremos el sistema, añadiendo el interfaz o librería que deberán utilizar los usuarios del mismo, al mismo tiempo que daremos algún ejemplo de Aplicación Distribuida que utilice las ventajas de nuestro sistema.

El autor

Jorge Delgado Mendoza es miembro de XFree86 desde el año 1994, como desarrollador responsable del servidor de Oak Technologies Inc. Es Ingeniero Superior de Telecomunicaciones por la UPM.

1. [3]El nombre significa *SERVER InterFace*, haciendo hincapié en su papel como punto de acceso de los clientes.

2. [4]En cualquier máquina distinta del líder, el valor de la inicialización del cerrojo es indiferente, ya que no se utiliza.

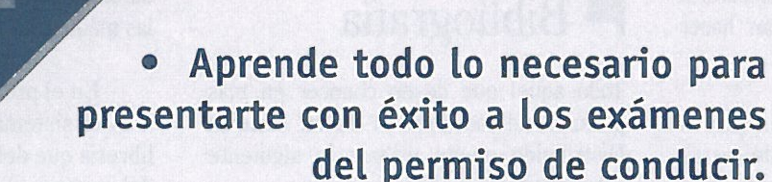
Curso completo de MECANOGRAFÍA PC MULTIMEDIA

P.V.P
2.995
Ptas. c.u.
IVA Incluido

- El método de este curso une el tradicional método 'ciego' con la gran efectividad de los ejercicios multimedia.
- Este curso te permitirá independizarte de los horarios fijos y llevar el aprendizaje a tu propio ritmo.

Curso completo de MECANOGRAFÍA PC MULTIMEDIA
2.995
Método probado que ofrece grandes resultados con un mínimo esfuerzo.
CURSO MULTIMEDIA PARA PRINCIPIANTES





- Aprende todo lo necesario para presentarte con éxito a los exámenes del permiso de conducir.
- Señalización, Seguridad vial, Normas de circulación... todo a tu alcance, de forma interactiva total.
- El método más eficaz para aprender sin esfuerzo.





Envíe este cupón por correo o fax (91) 661 43 86 o llamando al teléfono: (91) 661 42 11*
de lunes a jueves de 9:00 a 14:00 y de 15:00 a 18:00 h., y viernes de 9:00 a 15:00 h.

Deseo que me envíen: ☐ AUTOESCUELA MULTIMEDIA **por 2.995 ptas. + 250 de gastos de envío**
☐ MECANOGRAFÍA PC MULTIMEDIA **por 2.995 ptas. + 250 de gastos de envío**

Nombre y apellidos .... Domicilio ....
Población .... C.P. .... Provincia ....
Teléfono .... Edad .... Profesión ....

FORMA DE PAGO:

☐ Talón a ABETO EDITORIAL ☐ Giro Postal (adjunto fotocopia de resguardo)
☐ Contra-reembolso ☐ VISA nº ______ ______ ______ ______ Cad. __ __

ABETO EDITORIAL
C/ Aragoneses, 7

Simulación de sistemas (I)

Antonio José Novillo López (antonio@eicaiii.uc3m.es)

Introducción a la simulación

Muchas veces habremos oído la palabra simulación, sin entender plenamente su significado. Una definición formal podría ser: "La práctica de generar modelos para representar un sistema del mundo real o hipotéticos mundos futuros, experimentando con él para explicar el comportamiento del sistema, mejorar su funcionamiento o diseñar nuevos sistemas con características deseables". ¿Qué hay tras esta definición? Una actividad que, de una manera no profesional, todos realizamos día a día.

Cuando realizamos cualquier acción, normalmente la simulamos anteriormente de una forma mental como cuando movemos un mueble en una habitación. Antes nos imaginaremos el movimiento del mueble y cual serán las acciones mejores para realizar este movimiento. Pues esta acción tan habitual es lo que llamamos simulación.

Su utilización a nivel profesional se viene realizando desde hace aproximadamente medio siglo, auspiciado por el crecimiento de los ordenadores, que hacían posible la realización de muchas operaciones matemáticas, fundamentales en la simulación, y ha ido creciendo con el paso de los años, llegando a convertirse en una herramienta de uso fundamental en la industria e investigación, por una serie de ventajas frente a otros métodos tradicionales que enumeraremos más adelante.

Conviene precisar que la simulación no se restringe al uso de computadoras, y se puede aplicar de muchos modos, aunque el uso de éstas es el principal. Tomemos por ejemplo un túnel de viento, donde se prueban aviones, y todo tipo de material aerodinámico. En ese caso estamos simulando el comportamiento de, por ejemplo un avión, en una tormenta dentro de nuestro túnel de viento. Sin embargo, es mucho más sencillo e incluso económico reproducir en un ordenador los fenómenos físicos asociados al vuelo del avión y realizar la simulación en una computadora.

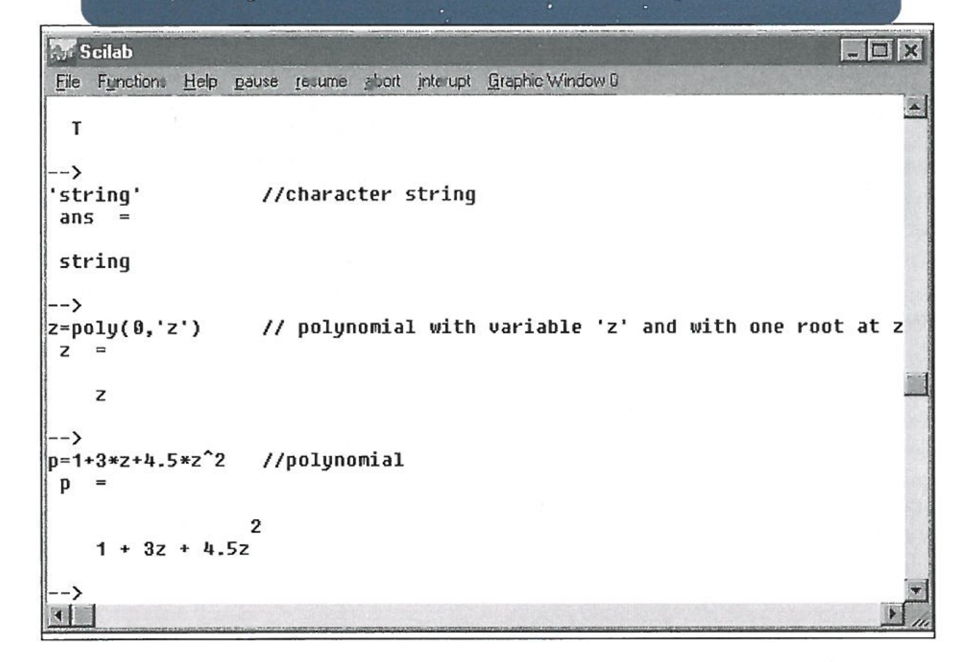
De todos modos, la simulación va unida fielmente al desarrollo de las otras ciencias, y no es posible simular aquello que no se puede estudiar adecuadamente, de modo que en algunas situaciones la simulación no es una situación posible.

Los sistemas simulados son de Entrada – Salida

Por otro lado debemos tener en cuenta que los sistemas simulados son de Entrada-Salida. Estos sistemas nos darán una salida para unos datos iniciales que nosotros debemos suministrar.

El impresionante aumento de la potencia de los ordenadores ha llevado a la simulación de sistemas y procesos a convertirse en una herramienta de gran uso dentro del mundo industrial. Aquí trataremos de ahondar en este tema.

Figura 1. Scilab, un entorno matemático similar a Matlab.



Por ello son incapaces de generar una solución por sí mismos. Sólo pueden servir como herramienta para el análisis del comportamiento de un sistema en condiciones especificadas por el experimentador.

La simulación se ha convertido en una herramienta de uso fundamental en la industria e invetigación

Algunos ejemplos de áreas donde la simulación es una herramienta de uso habitual podrían ser:

- Servicios financieros. Son habituales las simulaciones de sistemas bancarios o de seguros.
- Logísticas. Su uso es habitual en el diseño de almacenes o en el reparto de trabajo dentro de una planta de producción.
- Diseño en Ingeniería. Se suele simular el funcionamiento de circuitos electrónicos, complejos mecanismos y todo tipo de maquinaria. Nunca se produce un diseño sin antes haber sido simulado.
- Transporte. Se suele modelar y simular el tráfico en carreteras para descubrir posibles atascos o problemas de tráfico.

Ventajas e inconvenientes de la simulación

El hecho de que no siempre se pueda aplicar la técnica de la simulación y su uso no produzca una solución analítica, nos sugiere una pregunta: ¿cuándo es útil la utilización de la simulación?

En general, nos será útil siempre que nos sea más barato o más fácil que la realización del experimento sobre el sistema real, bien sea porque el experimento exige parar el sistema, no existen herramientas analíticas para desarrollar una solución o porque alguna de la condición a reproducir es difícil de conseguir.

En general, las ventajas que presenta la simulación son las siguientes:

- Permite, de una forma económica representar y estudiar prácticamente cualquier sistema, dentro de categorías científicas muy diferentes.
- El experimento se puede repetir tantas veces como sea necesario sin un gran coste adicional. Permite jugar con el tiempo de forma totalmente imposible para un experimento tradicional.
- Posibilita realizar experimentos que físicamente serían irrealizables.
- Permite explorar infinidad de alternativas para un problema sin modificar el funcionamiento del mismo, caso de una fábrica. Podríamos rediseñar el Layout sin necesidad de pararla y, de este modo, perder ingresos.

Pero no todo es bueno dentro de la simulación. Si tenemos fe ciega en ella podremos encontrarnos con problemas indeseables. Por ello es importante tener en cuenta sus inconvenientes:

- El desarrollo de un buen modelo de simulación es costoso y requiere de mucho tiempo.
- Puede parecer que una simulación refleja con precisión una situación de mundo real cuando, en verdad, no lo hace. Ciertos problemas intrínsecos a la simulación pueden producir resultados erróneos si no se resuelven correctamente.
- La simulación es imprecisa y no podemos medir con exactitud el grado de imprecisión.

Conceptos básicos

Las etapas en las que se dividirá un proceso de simulación podrían ser las siguientes:

1. *Definición del sistema*, determinación de los límites, restricciones y medidas de efectividad.

2. *Formulación del modelo*, reducción o abstracción del sistema real a un diagrama de flujo lógico o a un modelo matemático.
3. *Preparación de datos*, identificación de los datos que el modelo requiere y reducción de estos a una forma adecuada.
4. *Translación del modelo*, descripción del modelo en un lenguaje que comprenda el ordenador. En esta etapa será donde centraremos nuestros esfuerzos en los próximos números, donde aprenderemos a trasladar el modelo del lenguaje formal al lenguaje de simulación.
5. *Validación*, comprobar que el modelo realmente representa el sistema real.
6. *Planeación estratégica*, diseño de un experimento que producirá la información deseada. El experimento será lo que reproduzcamos con el ordenador. Una vez modelado el sistema fijaremos como varían sus parámetros y esto nos dará la información deseada.
7. *Planeación táctica*, determinación de cómo se realizará cada una de las ejecuciones de prueba especificadas en el diseño experimental.
8. *Experimentación*, ejecución de la simulación para generar los datos deseados y efectuar el análisis de sensibilidad.
9. *Interpretación*, obtención de inferencias con base en datos generados por la simulación.
10. *Implantación*, uso del modelo y resultados.
11. *Documentación*, registro de las actividades del proyecto y los resultados así como de la documentación del modelo y su uso.

Sistemas

- ¿Qué es un sistema?
Un sistemas puede ser definido como "una combinación de elementos o componentes interrelacionados entre si y con el global que actúan juntos para obtener un fin" (Natko). En este caso la palabra interrelación es el fundamento de la definición. Un sistema puede estar compuesto por uno o más subsistemas que a su vez también pueden estar divididos en otros subsistemas.

Una forma más sencilla de ver esta definición podría ser partiendo de un sencillo ejemplo. El sistema más grande es el Universo. Cada parte del universo que cortemos y de la que sepamos decir que pertenece a esa parte y que no será un sistema. Por tanto un sistema podrá ser prácticamente cualquier cosa. Un motor, una casa, un horno, etc. La clave está en poder diferenciarlo de su entorno.

- Clasificación de sistemas.
Generalmente los sistemas se clasifican en función de su estado. Definimos como estado de un sistema para un momento de tiempo al valor de un conjunto de variables que definen al sistema en dicho momento de tiempo. Según esto el sistema podrá ser estático o dinámico.

Entendemos por sistema estático aquel cuyos valores no cambian con el tiempo

Por tanto, *dinámico* será aquel cuyos valores varíen con el tiempo. Esta variación puede ser cíclica, puede ser aleatoria. Eso nos dará lugar a una nueva subdivisión dentro de los sistemas dinámicos. Esta clasificación se realizará en función al tipo de variables que conforman el sistema. Según este criterio existirán tres tipos de sistemas dinámicos: los continuos, los discretos y los híbridos, en los que las variables pueden ser tanto continuas como discretas.

Figura 2. Scicos, una aplicación gráfica para simulación.

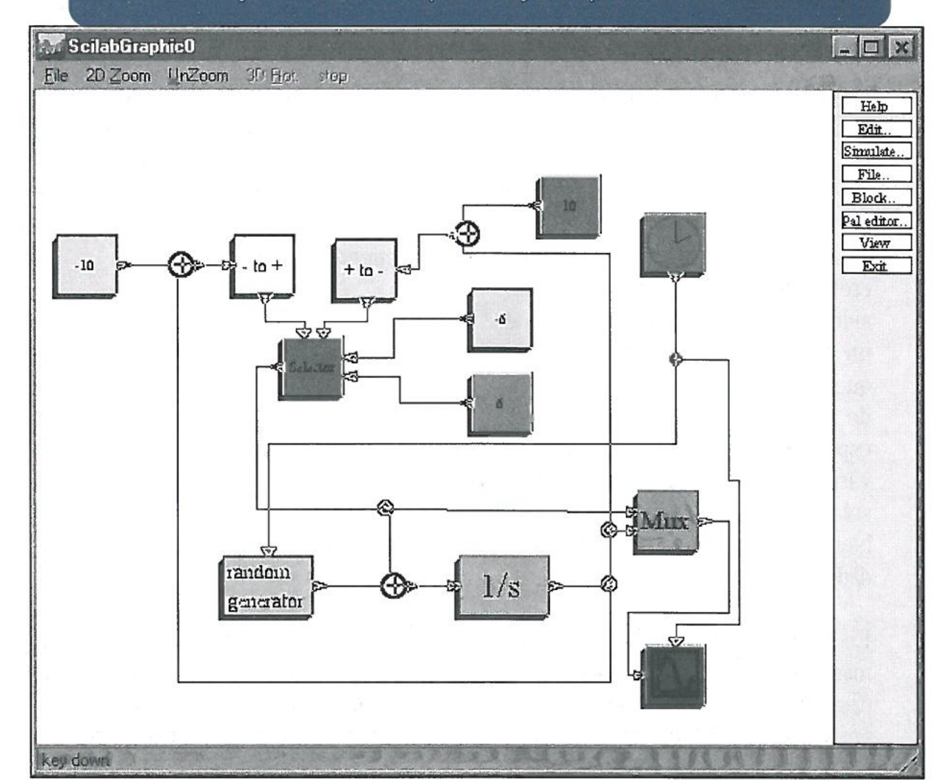
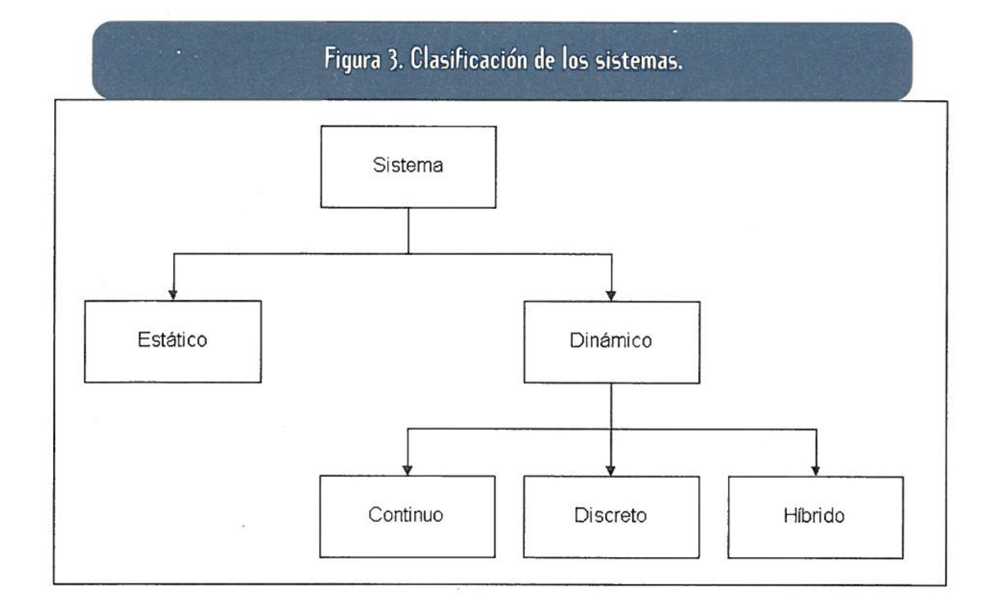


Figura 3. Clasificación de los sistemas.



encuentra en un estado *estacionario* cuando los cambios que se produzcan en el estado del sistema en el tiempo lo hagan dentro de un intervalo relativamente fijo. Por ejemplo, un avión que vuela en una dirección a una altura y velocidad fijadas previamente es un sistema dinámico, porque el avión se mueve, y además es estacionario. Un estado transitorio es aquel en el que se producen cambios bruscos en el estado del sistema. Por ejemplo cuando encendemos un fluorescente. Hasta que se fija la luz tenemos un periodo en que se producen variaciones. Durante ese tiempo el sistema se comporta de una forma *transitoria*.

Entendemos por sistema *continuo* aquel cuyas variables varían de forma continua en el tiempo. Un sistema continuo podría ser un péndulo, que se va moviendo y en cada momento de tiempo tiene una posición, que será la variable fundamental del sistema.

También lo sería un objeto caliente que se enfría en una habitación, por que el calor que va perdiendo va variando en cada instante hasta que alcanza el equilibrio, momento en que el sistema pasaría de dinámico a estático.

Un sistema dinámico se considerará *discreto* cuando las variables que le describen varíen de una forma discreta sobre el tiempo. Se puede considerar un sistema discreto la fila de un banco, tomando como variable que describe a dicha fila el número de personas que se encuentran esperando en ella. De este modo la variable será discreta porque varía solo en cantidades discretas. Nunca hay fracciones de clientes, sino unidades enteras.

En general, gran parte de los sistemas asociados a fábricas son sistemas discretos, porque los productos siempre varían en unidades discretas. Por ejemplo los paquetes que se cargan en un camión o los ladrillos que tienen que entrar en un horno, etc.

Un sistema en el que sus variables varíen unas de forma discreta y otras de forma continua se considerará un sistema *híbrido*. También lo será si sus variables son continuas pero se producen cambios en la causalidad del sistema. Esto es, que en determinados momentos varíen las ecuaciones que modelan el sistema. Un ejemplo sencillo sería una pelota que cae al suelo y rebota varias veces. Las ecuaciones que describen su posición son diferentes en el caso de la caída que en el del rebote hacia arriba. En un caso la fuerza de la gravedad está a favor del movimiento y en otro en contra.

Un sistema dinámico cambia en el tiempo de una forma aleatoria o cíclicamente

Otro concepto a tener en cuenta dentro del comportamiento de los sistemas dinámicos es si se encuentra en estado transitorio o estacionario. Diremos que un sistema se

Modelos

- Definición de modelo.
  Anteriormente hemos mencionado que un sistema es una sección de la realidad que es el foco primario de nuestro estudio. De cara a estudiar este sistema, primero deberemos realizar una representación del sistema. A esta representación es lo que denominamos *modelo*. Dicho de otra manera, realizaremos una abstracción del sistema que nos sea útil de cara a estudiar su comportamiento.

- Clasificación de modelos.
  Existen diversas posibilidades a la hora de generar el modelo de un sistema. En general, el modelo de un sistema va a consistir en un conjunto de ecuaciones o relaciones que nos permiten obtener los valores de salida del sistema respecto a unas variables de entrada.

  Una posible clasificación de los modelos puede ser:

  - Modelos *Físicos*: Como su propio nombre indica, son representativos de sistemas físicos. Son, dicho de un modo coloquial, maquetas de los sistema-

reales. Aproximaciones físicas al sistema real donde se realizan experimentos de cara a estudiar el sistema. No son de nuestro interés porque en ellos no aparece el ordenador.

- Modelos *Simbólicos*: Representaremos nuestro modelo mediante ecuaciones simbólicas. Pueden ser matemáticos y no matemáticos. Generalmente los matemáticos son los más utilizados por su consistencia, y unicidad, así como su relativamente fácil manejo. También existen modelos no matemáticos en que las relaciones están expresadas de modo gráfico (diagramas de flujo) o mediante sentencias lógicas. Nosotros nos centraremos, fundamentalmente en sistemas continuos, en los modelos matemáticos.

Un sistema continuo es aquel cuyas variables cambian de forma continua en el tiempo

Un ejemplo de modelo matemático podría ser la ecuación que representa el movimiento de un muelle excitado exteriormente.

+Kx=A sen t

x representará la posición del muelle en cada momento. Resolviendo esta ecuación diferencial conoceremos la posición del muelle en cada instante de tiempo. En este caso la solución es relativamente sencilla de calcular y no tiene mucho interés resolverla mediante una simulación porque analíticamente podemos calcular la solución de la ecuación, que será una combinación de senos y cosenos.

- Validación de modelos.
  Cualquier modelo no es válido para representar un sistema. Deberemos utilizar un método que nos permita averiguar cuan parecido es nuestro modelo a la realidad de cara a evitar lamentables errores que puedan echar a perder nuestra simulación.

Aquí es donde debemos darnos cuenta de la importancia del modelo. Cualquier valor que obtengamos será nulo si el modelo no se comporta como el sistema. De hecho, esta es la parte más complicada y tediosa del proceso de simulación. Es más. algunos sistemas extremadamente complicados son muy difíciles de modelar y los científicos pueden tardar años en conseguir un modelo, que en muchos casos sólo es válido para un pequeño rango de valores. Un caso típico es la mecánica de fluidos y también los procesos de combustión.

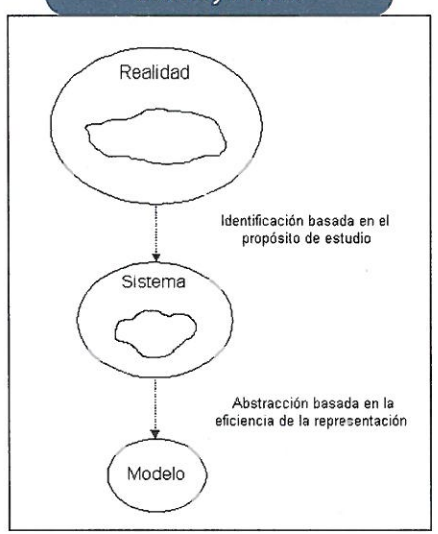
Simulación

Tras haber estudiado con detenimiento qué es un sistema y cómo se modela, llegamos a la parte que más nos interesa: la simulación. Ésta es realmente la que usa el ordenador como herramienta principal y es la que vamos a intentar desarrollar.

Una definición más académica de la simulación sería: "Es la técnica de construir y poner en funcionamiento el modelo de un sistema real con la intención de estudiar su comportamiento sin irrumpir en el entorno del sistema real" (Koskossidis y Brennan).

De una manera más llana, la simulación será el proceso de encontrar las soluciones a las ecuaciones que modelan nuestro sistema para unos valores iniciales que nosotros fijaremos. Posteriormente, con todo ese conjunto de datos obtenidos en la simulación, realizaremos las operaciones necesarias de tratamiento de datos estadístico y gráfico

Figura 4. Relación entrés realidad, sistema y modelo.



para acomodar los datos para su posterior estudio. Muchas de las herramientas de simulación incluyen potentes programas gráficos que nos muestran como salida la variación o relación entre datos, pero hay que tener en cuenta que esto solo es una aplicación más del programa, pero no del proceso de simulación. La simulación únicamente nos dará una lista de valores para las variables que definen el estado del sistema. El uso posterior de esos datos ya no pertenece al proceso.

Herramientas de simulación

En general, el proceso de simulación se puede realizar de múltiples maneras. En primer lugar, debemos crear nuestro modelo matemático en un lenguaje que el ordenador pueda entender. Seguidamente deberemos resolver este modelo usando, por lo general, algoritmos de cálculo numérico.

Como se comentó con anterioridad, el uso de la simulación es especialmente útil en sistemas cuya solución analítica es difícil de conseguir. Por ello, al simular utilizaremos métodos numéricos de cara

Figura 5. La simulación consiste en la resolución experimental del modelo.

Modelo

Clasificación de sistema: Estático — Dinámico

Método de solución: Analítico — Experimental — Analítico — Experimental

Simulación

a obtener una solución al conjunto de ecuaciones definitorias del sistema.

Cualquier modelo no es válido para representar un sistema

El proceso anteriormente descrito se puede realizar mediante diversas herramientas. Unas son generales, como pueda ser la programación tradicional o el uso de entornos matemáticos de carácter general y otras son específicas para simulación, como sería el lenguaje ACSL o Simulab. A continuación vamos a describir con un mayor detalle algunas de estas herramientas.

Programas de uso general

- Programación tradicional: C, Pascal, Fortran.
  Tradicionalmente se ha utilizado mucho el fortran como lenguaje para aplicaciones científicas. Gran parte de los algoritmos utilizados en ingeniería han sido realizado sobre fortran. Sin embargo, cualquier lenguaje es bueno para la simulación, siempre que disponga de librerías matemáticas. De hecho, incluso el Basic más tradicional se utiliza en algunos casos. Nosotros, de ahora en adelante, los ejemplos que realicemos los haremos en C, que es un lenguaje muy extendido y de gran potencia y versatilidad.

  La metodología de simulación mediante este tipo de programación es sencilla. Si el sistema es sencillo, calcularemos la solución analítica y a partir de ella iremos resolviendo el sistema de ecuaciones. Si no podemos resolver analíticamente, deberemos implementar un algoritmo de resolución de ecuaciones numéricamente o utilizar alguna librería de libre distribución, de las que hablaremos posteriormente, que disponga de estos algoritmos.

  La gran desventaja de la programación tradicional es que tenemos que hacer un programa completo para cada simulación y que no es fácil describir el sistema mediante un lenguaje tradicional.

- Entornos matemáticos,como los de Mathematica, Matlab, Scilab,...etc.
  Estos son entornos para uso matemático general. Matlab y Scilab están fundamentalmente orientados, especialemente Matlab, al manejo de expresiones matriciales. Matlab es posiblemente uno de los programas más utilizado por los ingenieros de todo el mundo, fundamentalmente por la riqueza de librerías que se distribuyen para el programa y que permiten utilizarlo en prácticamente todos los campos de la ingeniería. Scilab es un programa de características similares, más orientado a la ingeniería de control y de sistemas, que está desarrollado por un instituto de investigación francés, INRIA, y que es de libre distribución para uso no comercial y uso académico. Mathematica está más orientado a las matemáticas, pero también es un programa muy potente que se usa en otras disciplinas.

  Todos estos entornos tienen su propio lenguaje, de muy alto nivel, donde describiremos mediante ecuaciones o matrices nuestro sistema y lo resolveremos con las rutinas internas que incluyen los propios programas.

Estos entornos están fundamentalmente orientados al manejo de expresiones matriciales

Como ventaja frente a la programación tradicional encontramos la facilidad de resolver las ecuaciones y el fácil manejo de los resultados posteriores. Sin embargo, el modelo del sistema tampoco es fácil de representar, aunque más que en el caso anterior.

Programas específicos

Los entornos específicos suelen tener todos una estructura muy similar. Desarrollan un lenguaje propio de muy alto nivel con un analizador sintáctico y

mediante el cual es muy fácil describir el modelo y luego hacen uso de unas librerías de libre distribución con algoritmos numéricos, como puede ser Linpack, con las que resuelven el sistema de ecuaciones que define el modelo.

Los entornos específicos desarrollan un lenguaje de alto nivel que facilita la descripción del modelo

- Lenguajes específicos: ACSL, CCSL.
  Son lenguajes de muy alto nivel que siguen la estructura anteriormente descrita. Su uso es cada vez menos habitual, pero han sido los reyes de los años 80. El uso de las herramientas gráficas les está quitando popularidad. Se basan en un lenguaje muy sencillo mediante el que describir el modelo que posteriormente pasan a resolver.

- Entornos gráficos: Simulab, Scicos, Dimola.
  Son las alternativas más modernas y cómodas. Quizás complicadas para grandes simulaciones, son excelentes para los principiantes. Utilizan un entorno gráfico, casi visual para crear el modelo, lo resuelven y nos dan una salida gráfica. Cada elemento del modelo será una caja a la que podremos realizar operaciones de todo tipo. Las variables irán pasando de caja en caja. Podremos integrarla, derivarla, sumar, restar, pasarle un filtro, etc.

  La gran ventaja de este tipo de productos es su manejo intuitivo y muy similar a los diagramas de bloques tan usados a la hora de modelar. En general, suelen trabajar sobre entornos matemáticos que aportan los algoritmos y el lenguaje que va por debajo del entorno gráfico. Simulab es un programa de la misma casa que Matlab. Scicos se distribuye junto a Scilab con la misma licencia. Dimola es un programa independiente y de gran proyección. Posee asimismo un lenguaje descriptivo que permite que algunas partes las realicemos en dicho lenguaje y otras gráficamente.

- El futuro: Modellica.
  Modellica no es un lenguaje de simulación sino de modelado. Se trata de un estándar en fase de ser aprobado y que pretende uniformizar los lenguajes de modelado. Se pretende con ello que en un futuro todos los entornos de simulación usen Modellica como lenguaje de modelado. De este modo el modelo podrá portarse de unos simuladores a otros sin necesidad de adaptar el modelo a cada entorno. Todavía se encuentra en fase de estudio y pasarán algunos años hasta que se acepte, pero es, sin duda, el futuro del modelado.

En un futuro, con el uso de Modellica, no será necesario adaptar el modelo a cada entorno

- Las herramientas para los campos específicos:
  Algunos campos, como pueden ser el desarrollo de circuitos o el cálculo de estructuras disponen de herramientas de simulación propias. En el caso de los circuitos destaca sobre todo el SPICE, un lenguaje de descripción y simulación de circuitos con una muy alta implantación en el mundo de la electrónica y la electricidad. Existen otros lenguajes también, como VHDL, para descripción de circuitos lógicos.

  En el campo del cálculo de estructuras destaca sobre todo el uso de elementos finitos, una técnica de resolución de estructura que permite calcular las deformaciones y tensiones a las que se va a ver sometida una pieza. Muchos son los programas que hacen uso de esta técnica, siendo COSMOS uno de los más conocidos.

Resumen

A lo largo de este artículo hemos visto, a grandes rasgos, en que consiste la metodología de la simulación, una herramienta de uso cada vez más generalizado dentro de los más diversos campos científicos. Hemos descrito cada una da las actividades a realizar de cara a realizar una simulación, poniendo especial hincapié en la importancia de crear un buen modelo del sistema a estudiar de cara a su posterior simulación. También hemos descrito brevemente las distintas herramientas de simulación que poseemos en este momento.

En el próximo número entraremos ya de lleno en la realización de nuestra propia simulación sobre un sistema continuo mediante los diversos métodos descritos en este artículo.

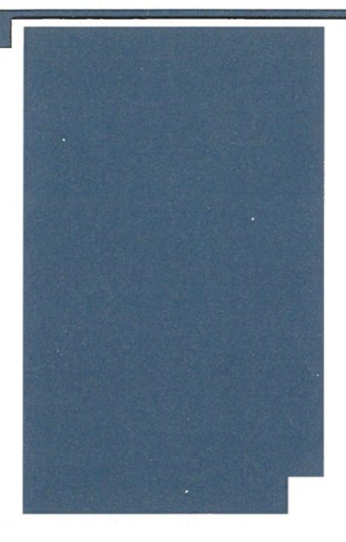
Referencias

Para obtener una información más completa y extensa sobre los sistemas de simulación mirar la siguiente bibliografía:

- Simulación de sistemas
  Robert E. Shannon
  Ed. Trillas

- Simulation and Modelling of Continuous Systems
  Matko, Karba, Zupancic
  Ed. Prentice Hall

- Continuous Systems Modelling
  Francois E. Cellier
  Ed. Springer-Verlag



Correo del lector

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

Pregunta

Hola, soy un suscriptor de Sólo Programadores. He leído vuestro artículo del último número y me ha solucionado, en parte, la vida, de manera que desde ya os agradezco vuestro trabajo en esta gran revista.

Después de las felicitaciones me queda lo importante, allá voy. Tengo una línea RDSI o ISDN (aquí en Luxemburgo donde yo vivo se llama ISDN), con su correspondiente módem (US Robotics) instalado en un servidor NT 4.0. Tengo entendido que al ser un "router" podría conectar a Internet a través de los ordenadores de la red (son sólo dos, la red está en mi casa) y me gustaría navegar desde el ordenador con el que trabajo habitualmente, que tiene la tarjeta gráfica más potente, la de sonido, ...y no a través del servidor, que lo "único" que tiene son discos duros. En fin, os agradezco de antemano toda la ayuda que me podáis prestar. Gracias!!

Respuesta

Bien. La solución a tu problema no tiene mucha complicación. Existen varias maneras de habilitar el enlace, pero la más rápida y sencilla consistiría en instalar un programa Proxy en el servidor de discos NT 4.0, que filtre todas las comunicaciones de los puertos TCP/IP y las redirija a uno u otro ordenador de la red según de dónde haya venido la petición de la información. En el CD-ROM de este mes podrás encontrar una excelente aplicación que actúa como Proxy tanto para Windows 95 como para NT 4.0 llamada WinGate cuya versión actualmente es la 2.0. En el ordenador cliente lo único que tendrás que hacer será indicar en todas aquellas aplicaciones Internet, la dirección URL del servidor Proxy que hayas configurado en NT 4.0 para que todas las comunicaciones se realicen a través de él.

No creemos que vayas a tener ningún problema en su instalación pues incluye una ayuda muy completa y su configuración es casi automática. Recomendamos encarecidamente su uso para tal fin, ya que en este campo existen varias aplicaciones para lograr nuestro objetivo y ésta es, sin duda alguna, la más compatible con la mayoría de estándares o protocolos de Internet (RealAudio, FTP, Telnet, ...).

Pregunta

Hola amigos. Soy un lector de su revista y les escribo para ver si me pueden solucionar la siguiente pregunta. Estoy programando una sencilla aplicación para Visual Basic 5.0. (muy sencilla puesto que estoy empezando) y me gustaría que se ejecutase también bajo el Windows 3.1. No he encontrado ninguna herramienta o utilidad para pasar código del Visual Basic 5.0 a Visual Basic 4.0 16 bits, así que la he reprogramado bajo este último. Funciona perfectamente pero no consigo que la ventana de la aplicación de la versión de 16 bits aparezca centrada en la pantalla independientemente de la resolución que se tenga configurada. ¿Qué parámetro o qué tengo que hacer en VB4 para conseguir esto?

Respuesta

Cierto. En Visual Basic 5.0 hay un parámetro en las Propiedades de la aplicación exclusivamente para conseguir el centrado de la ventana. Esta propiedad de ventana no se encuentra disponible en Visual Basic 4.0, ya que para conseguir este efecto, deberemos introducir la siguiente línea en el código fuente de la ventana:

Me.Move (Screen.Width - Me.Width) / 2, (Screen.Height - Me.Height) / 2

Este mes hemos realizado un esfuerzo especial para ofreceros como CD-ROM adicional RED HAT LINUX 5.0. Esperamos que sea de vuestro agrado.
En cuanto a los pasos a seguir para su instalación, hemos incluido en el CD-ROM correspondiente a Sólo Programadores, un apartado específico que contiene el manual en línea del producto así como el proceso de instalación en castellano.
El directorio que contiene la información es UNIX\REDHAT50.
Además se incluye un listado de direcciones Web para ampliar información.



Exchange Server 5.5, DB2 5.0, WordPerfect para Linux 7.0 y ICQ98a

Microsoft Exchange Server 5.5

Una plataforma de mensajería no es nada sin una buena base. Esta base tiene que poseer unos fundamentos sólidos, escalables, fiables y seguros que ofrezcan un gran rendimiento y disponibilidad y que estén respaldados por un conjunto de servicios y soporte que puedan gestionar las necesidades del departamento y empresa.

Microsoft Exchange 5.5 permite elegir los protocolos y los clientes que se van a usar sin encasillarse en un sólo protocolo. Coexiste con otros sistemas de correo y cumple los estándares OSI. Provee también de conectividad en sistemas basados en host como OfficeVision/VM (PROFS) and SNADS, y sistemas basados en LAN como Notes, MS® Mail y cc:Mail. Aún mejor, Exchange puede actuar como pasarela entre ellos.

Integra herramientas de colaboración en grupo que exprimen al máximo las capacidades de la plataforma de correo. Para ello se puede hacer uso de diversas utilidades con las que el usuario esté familiarizado, aprovechando los conocimientos de la propia herramienta y su experiencia en ella.

El entorno de desarrollo nos permite construir aplicaciones que proporcionen acceso a la información mediante un cliente estándar, tal como un navegador de red. Y lo que es más, esta plataforma también nos suministra componentes de valor añadido, como pueden ser servicios de chat en tiempo real.

En este número hemos incluido todo el paquete disponible ofreciendo el servidor, los conectores, el software de soporte con todo tipo de utilidades y la última versión del Outlook97 tanto para Windows NT, que es donde se instala el servidor, como para Windows 95.

DB2 Universal Database 5.0

Última entrega de la familia DB2 de productos sobre bases de datos relacionales. Ofrece manejo de bases de datos de calidad industrial para toma de decisiones, proceso de transacciones, y un extensivo rango de aplicaciones de negocios. La familia DB2 se extiende a sistemas AS/400, hardware de tipo RISC System/6000, Mainframes IBM, máquinas no-IBM del tipo Hewlett-Packard y Sun Microsystems, y sistemas operativos como OS/2, Windows (95&NT), AIX, HP-UX, SINIX, SCO OpenServer, y entorno operativo Solaris.

El producto ofrece 60 días de prueba desde su instalación. La versión aquí presentada es la de Windows 95 y NT, tanto servidor como clientes. Además IBM proporciona una completa documentación navegable en el CD ya que está en formato HTML.

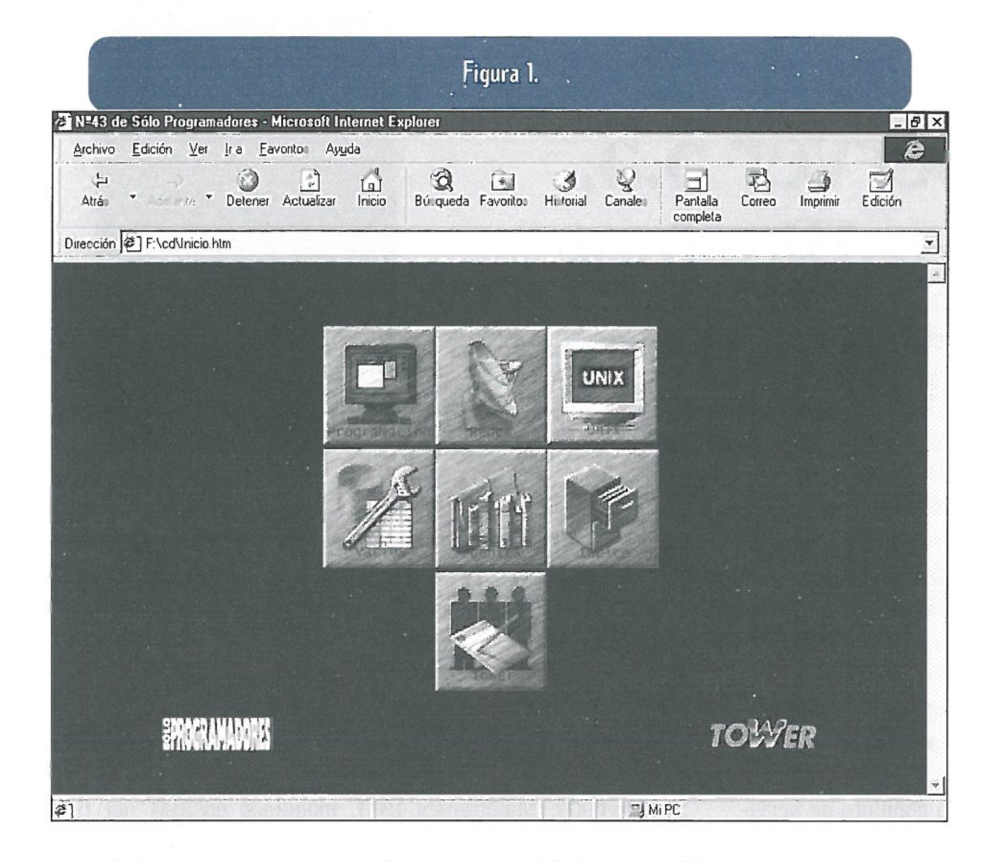
WordPerfect para Linux 7.0

Versión 7.0 del famoso WordPerfect para Linux en su versión de evaluación. Caduca a los 15 días a partir de la instalación del producto. Esta edición está completamente integrada con las X Windows e incorpora un manual de ayuda de lo más completo. Además, hemos incluido dos paquetes de lenguaje, el correspondiente al inglés y al español.

ICQ 98a

Con más de tres millones de usuarios registrados y más de siete millones de usuarios totales, ésta es la herramienta más potente de comunicación e interconexión entre usuarios de Internet.

Figura 1.



Este programa arranca un icono en la barra de inicio, el cual se activa en cuanto se conecta el ordenador a una red TCP/IP como Internet. Se da de alta (en este último caso) en el servidor ICQ haciéndonos visibles como que estamos conectados en ese momento, al resto de los usuarios de la red que tuvieran nuestro número personal #ICQ dado de alta en su programa cliente y permitiendo

Figura 2.

así chatear con nosotros, intercambiar mensajes urgentes, E-Mail y arrancar clientes externos.

Esta última opción es la más potente, puesto que podemos configurar programas para lanzar desde el ICQ tales como el Netmeeting, Quake, programas de telefonía, y todo tipo de aplicaciones que funcionen bajo Internet/intranet.

Es muy fácil de integrar en nuestras páginas Web personales. Dispone de paneles insertables que permiten automatizar el proceso de intercambio de números ICQ, además de ofrecer al lector de nuestra página la opción de saber si estamos conectados en ese momento, aunque no disponga del correspondiente programa cliente.

Software para Red de Network Associates (McAfee)

NetXRay analiza y monitoriza cada segmento de la red en una organización, incluso en pequeñas redes locales o en ramas de oficinas remotas. El analizador NetXRay es una herramienta de gestión de rendimiento que captura datos, monitoriza el tráfico de la red y muestra las estadísticas claves de la misma.

Esta utilidad facilita a los administradores de red extraer información vital para la solución de problemas, mejora del rendimiento. Además de proporciona un experto asesoramiento a la hora de migrar los actuales y complejos entornos de red.

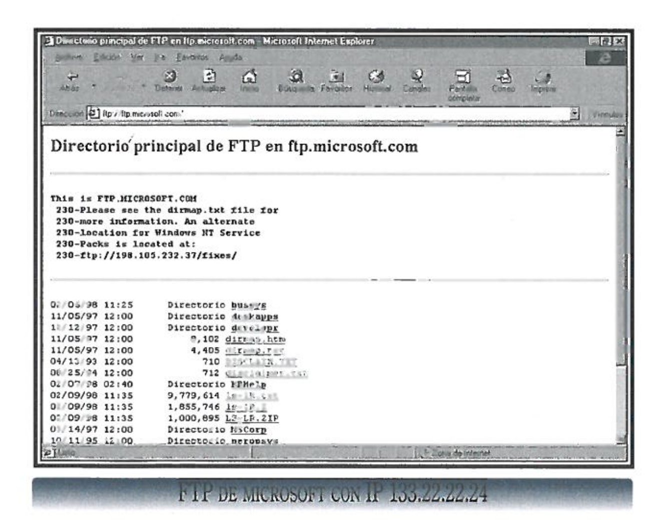
Además de esta herramienta incluimos una suite de análisis de redes en entornos distribuidos que es capaz de comprobar por separado el funcionamiento de cada componente de la red, desde un Hub al último Switch.

8 INTERNET Y SUS TECNOLOGÍAS (II)

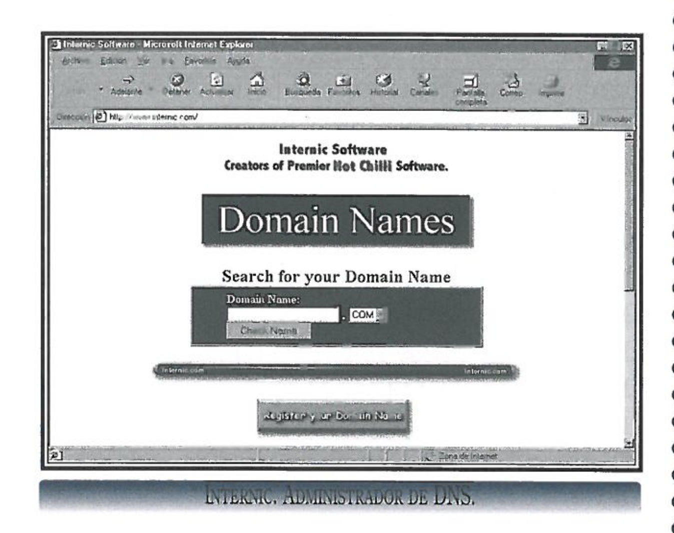
RESOLUCIÓN DE NOMBRES ->DNS

El sistema de nombres de dominio (DNS) es lo que nos permite que al navegar por Internet no tengamos que conocer las direcciones IP para llegar a los servidores WEB (o a cualquier otro recurso) que queremos visitar. Es un conjunto de protocolos y servicios que permite a los usuarios utilizar nombres para buscar otros host. Por ejemplo, supongamos que si la dirección IP del sitio FTP de microsoft es 133.22.22.44, la mayoría de la gente llega a este equipo especificando ftp.microsoft.com y no la dirección IP que es más difícil de recordar. Otro motivo para usar DNS es que, si queremos cambiar la dirección IP de una máquina y que el dominio continúe apuntando a la misma máquina, lo podremos hacer sin problemas. Sólo tenemos que comunicar la nueva dirección al servidor DNS. El nombre ftp.microsoft.com apuntará a otra dirección IP pero nadie se habrá dado cuenta del cambio.

Otro modo de resolver los nombres de dominio a direcciones IP es utilizando un archivo HOST. Es decir, un archivo que tenemos en nuestra máquina y en el que están contenidos



FTP DE MICROSOFT CON IP 133.22.22.24



INTERNIC. ADMINISTRADOR DE DNS.

todos los nombres de dominio y sus direcciones IP equivalentes. Esto es válido para una red interna que está usando TCP/IP, pero no para Internet ya que el tamaño de éste fichero sería inmenso y no se modificaría dinámicamente cada vez que un cambio fuese necesario.

Jerarquía DNS

Un servidor DNS consta de una base de datos de nombres. Los nombres de esta base de datos establecen una estructura de arbol. Cada nodo o dominio del arbol tiene un nombre y puede contener otros nodos o subdominios. Los nodos se agrupan en zonas para hacer posible el distribuir la administración. Los host representan las hojas del arbol. De éste modo una dirección DNS como saludo.holaatodos.es representa a un host llamado saludo que pertenece al dominio holaatodos.es y para acceder a el tendremos que escribir su nombre al completo.

INTERNIC

Internic (Internet Network Information Center, http://www.internic.com) es el centro de información de la red de internet que administra la

raiz de la base datos DNS en Internet. Los dominios superiores se han asignado a organizaciones y países como se muestra en la siguiente lista:

- ❑ com Comercial
- ❑ edu Educacional
- ❑ gob Gubernamental
- ❑ int Organizaciones internacionales
- ❑ mil Operaciones militares
- ❑ red Organizaciones de redes
- ❑ org Organizaciones no comerciales

De este modo uned.edu es utilizado para la universidad de educación a distancia.

Software necesario

Microsoft DNS sirve para utilizar la resolución de nombres en sistemas basados en Windows NT. Además para NT tenemos la ventaja de que con Microsoft DNS podemos integrar la resolución de nombres con WINS.

FTP

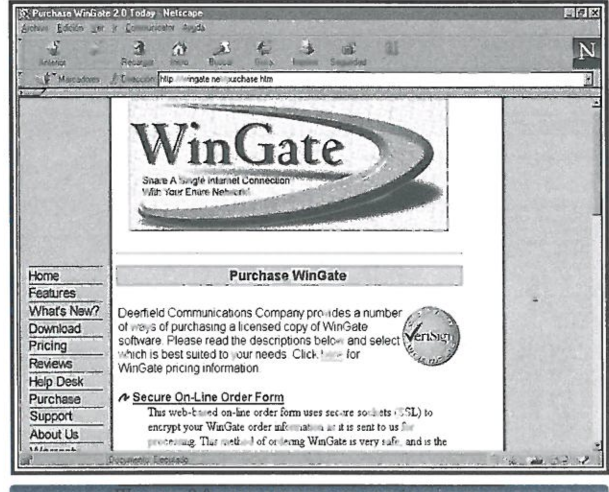
Como su nombre indica, File Transfer Protocol FTP es un protocolo o método, acordado de entendimiento entre los programas complementarios de dos máquinas, que tiene por objeto el intercambio de archivos.

Con Windows NT 4.0 y UNIX los servidores FTP se entregan con el sistema operativo. El servidor FTP tiene un uso fundamental en las redes internas y en las Intranet's como almacén y fuente de los archivos más utilizados. El administrador del sistema crea una cuenta especial denominada *anonymous* que puede ser utilizada por cualquier usuario, pero que sólo la tienen aquellos hosts de Internet que hayan configurado este servicio.

PROXY

Si queremos dar acceso a Internet a una red local, utilizando una sola cuenta de conexión a Internet, tenemos que utilizar un Proxy. Un Proxy es un software capaz de recibir peticiones de páginas WEB desde una red local y lanzarlas a Internet. El proxy va a hacer de canal de paso a todas las salidas a Internet desde la red local. Las direcciones de la red local no son válidas en Internet, pero el ordenador que tenga instalado el proxy ha de tener una dirección IP válida dentro de ella (la obtendrá al conectarse a Internet).

Podemos instalar un proxy en un ordenador en el cual tengamos configurado el acceso telefónico a redes. Conectaremos este ordenador a Internet utilizando un módem o una línea RDSI. El Proxy recibirá peticiones de páginas Web de Internet desde direcciones IP de la red local y realizará la petición solicitada, utilizando la dirección IP que es válida dentro de Internet. El Proxy apunta quién le ha realizado las peticiones desde la red local y, cuando posee la respuesta, ésta se la envía a sus clientes.



WINGATE 2.0 ES UNO DE LOS PROXIS MÁS CONOCIDOS.

Funciones del Proxy:

- Almacenar las páginas Web recibidas por si los usuarios se las vuelven a solicitar, de este modo no tiene que volver a pedirlas. Gracias a esto, la velocidad de navegación para la red interna aumenta considerablemente.
- Se puede configurar para restringir las posiblidades de la navegación de la red interna.
- Mantenimiento de un fichero de log's en el que se apuntan todas las páginas que han sido visitadas desde la red interna y quién las ha visitado.

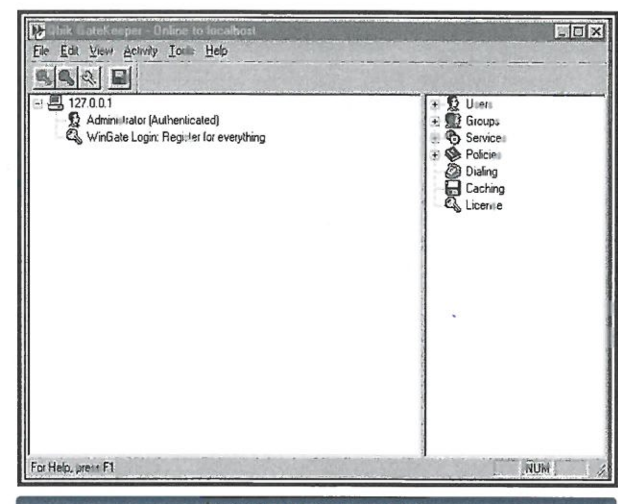
WinGate 2.0

WinGate 2.0 es un servidor Proxy/Firewall disponible bajo Windows 95 o Windows NT. Comparte una única conexión Internet (módem, RDSI, LAN) a través de toda una red y permite a varios usuarios de la red navegar por Internet, bajarse ficheros, manejar E-mail y mucho más. También actúa como una potente aplicación cortafuegos de seguridad, implementando SOCKS5, encriptación, grupos de usuarios, servicios de seguimiento, etc.

La versión incluida en el CD viene con licencia para un usuario. Si se planea permitir acceder a Internet a través de WinGate a más de una máquina a la vez, necesitaremos registrarnos en función del número de máquinas que deseamos accedan simultáneamente.

Instalación de WinGate

Para instalar el servidor WinGate ejecutamos el archivo wg20bNT.exe o wg20b95.exe. Al ejecutar uno de estos archivos instalaremos

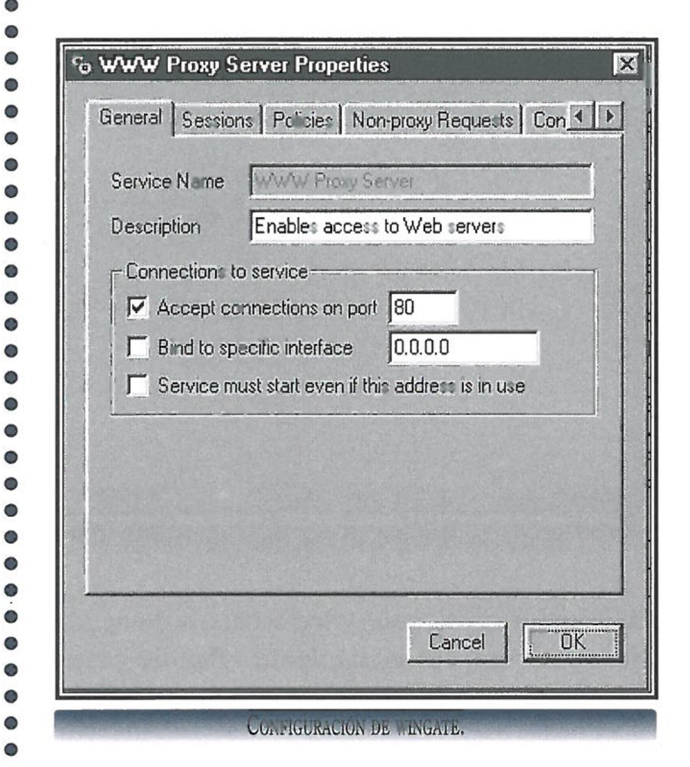


ADMINISTRADOR DEL WINGATE.

WinGate, GateKepeer y los archivos de ayuda. Se creará entoces el siguiente árbol de cuatro directorios que se especifica a continuación:

c:\Program Files\WinGate\logs
c:\Program Files\WinGate\audit
c:\Program Files\WinGate\cache
c:\Program Files\WinGate\resources

- Audit -> Es el directorio donde se guardarán las auditorías que se le hagan a los usuarios. Podemos programar Wingate para que, cuando un usuario realice determinado evento, éste se anote en este directorio.
- Logs -> Los archivos de logs referentes a la navegación de la red interna (Se anotan los sitios que han sido visitados desde la red interna).
- Caché -> La caché de Wingate, es el lugar donde se almacenan páginas HTML que han sido recientemente solicitadas por el usuario.
- Resources -> Se refiere a los diferentes recursos como los gif´s transparentes, etc.



CONFIGURACIÓN DE WINGATE.

Configuración y control de WinGate.

Para configurar y controlar WinGate se utiliza una herramienta de configuración llamada GateKeeper. Los usuarios y grupos de usuarios van a ser utilizados de manera similar a como se hace en Windows NT con el administrador de usuarios. El paradigma de usuarios-grupos es el mismo.

Lo primero que nos aparece al abrir esta herramienta de configuración es una pantalla para introducir la clave. Si introducimos una clave válida ya podemos utilizar GateKepeer.

La pantalla se divide en dos apartados completamente diferenciados. El frame de la izquierda es el apartado en el que se muestra la actividad que está pasando por el proxy. En él, podemos ver por ejemplo cuáles son las páginas HTML que están siendo visitadas desde la red interna en este mismo instante. El apartado de la derecha es el frame de configuración.

En el frame de configuración podemos tener hasta ocho ramas colgando de la raiz del árbol principal:

- Usuarios. Podemos crear y borrar usuarios. Al igual que los usuarios de Windows NT, un usuario puede pertenecer a un grupo y tener una serie de derechos. Podemos configurar a un usuario para que, mediante una determinada contraseña, el también tenga permisos para configurar WinGate. Podemos determinar qué aspectos de un usuario queremos auditar, etc.
- Grupos. Dependiendo de a qué grupo pertenezca un usuario, así serán sus derechos. Los grupos por defecto son administradores, usuarios y dial-in', cada grupo tendrá derechos de acceso diferentes. Los usuarios pueden pertenecer a uno, varios, todos o ningún grupo.
- Localidades. En la versión completa tenemos visible en esta rama del árbol a los ordenadores conectados a Wingate desde la red interna.
- Servicios. Aquí es donde podemos ver cuál es el potencial de WinGate. En esta rama del árbol vamos a configurar cuáles son los servicios que queremos desempeñar con el proxy. Podemos configurar los servicios que vamos a hacer accesibles para la red interna. Telnet, Ftp, WWW, IRC, etc.
- Políticas. En este apartado se configuran los derechos de acceso al sistema y las políticas con los permisos.
- Dialer. Las propiedades del dialer.

- Caché. Opciones de configuración de la caché: tamaño, cuándo utilizarla, etc.
- Licencia. Para que poder identificarse como usuarios registrados.

Requerimientos

LAN: 2-5 usuarios
486 DX2/66 8Meg RAM o superior
Windows 95 o Windows NT
28k8 modem
TCP/IP instalado
WinGate 2 Lite o Pro

LAN: 5-20 users
486 DX2/66 20 Megas de RAM o superior
Windows NT
Conexión ISDN
TCP/IP instalado
WinGate 2 Lite o Pro

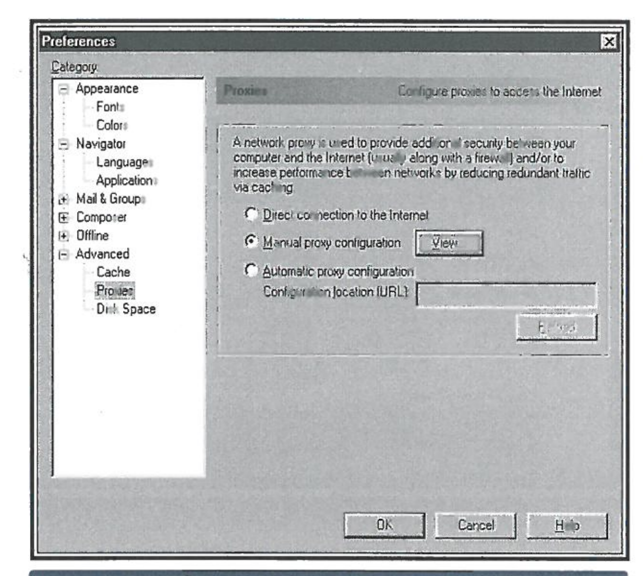
LAN: 20 + usuarios
Pentium 90 con 32 +Megas de RAM
Windows NT
Conexión ISDN T1
TCP/IP instalado
WinGate 2 Lite o Pro

Instalación en los clientes

Para que un navegador de la red interna pueda utilizar Wingate, hemos de decirle cuál es el ordenador de la red que tiene instalado el software de proxy.

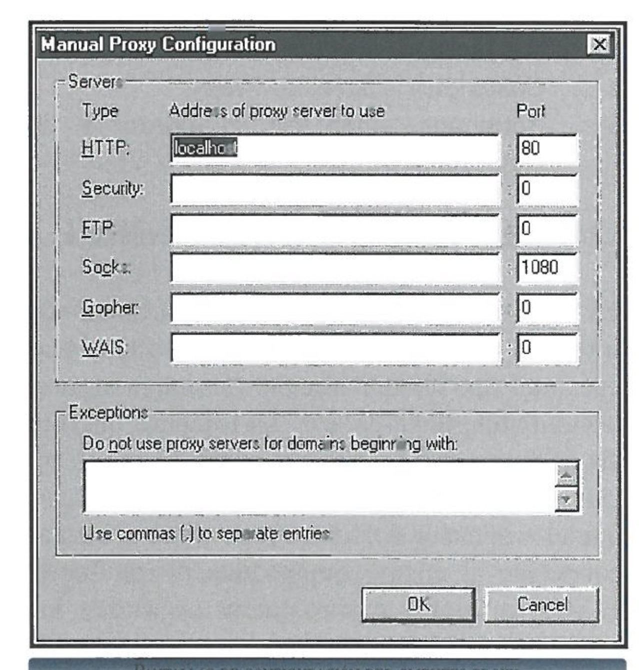
Netscape.

1- Desde el navegador elegimos la opción Edit -> Preferencias.
2- Avanzadas -> Proxies.
3- Configuración manual de Proxies.

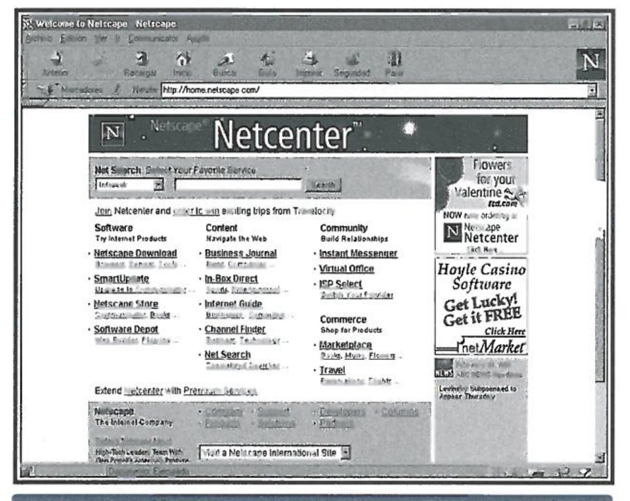


CUADRO DE DIÁLOGO DE PREFERENCIAS.

4- Elegir, para cada uno de los servicios que tenemos disponibles, la dirección del ordenador que hará de proxy para estos servicios y en qué puerto estará escuchando. En el campo dirección, tendre-



PANTALLAS DE CONFIGURACIÓN DEL CLIENTES PROXY.



NESCAPE OFRECE SERVICIOS DE TODO TIPO.

mos que introducir la dirección ip del ordenador que tiene instalado Wingate o su DNS. En el puerto, tendremos que introducir el puerto para el servicio. Por defecto para HTTP es el 80.

Explorer.

1- Desde el menú del navegador elegimos Ver -> Opciones.
2- Conexión -> Servidor Proxy.
3- Actuamos como en el punto 4 de Netscape.

CORREO ELECTRÓNICO: E-MAIL

Es uno de los recursos más importantes de Internet y gracias a él, se puede transportar cualquier tipo de información, ya sean mensajes, documentos, ficheros, etc. La principal diferencia de este servicio, respecto a los demás servicios de Internet (WWW. Telnet, FTP, etc.), es que el ordenador destinatario no tiene que estar conectado al mismo tiempo que el remitente. Para las máquinas por las que se encaminan los mensajes, es necesario conocer para quién es el mensaje que están transmitiendo sin que sea necesario ver su contenido. De esta forma toda la información necesaria para que el mensaje alcance su destino se encuentra en lo que se denomina "cabecera" y el contenido del mensaje en el "cuerpo del mensaje".

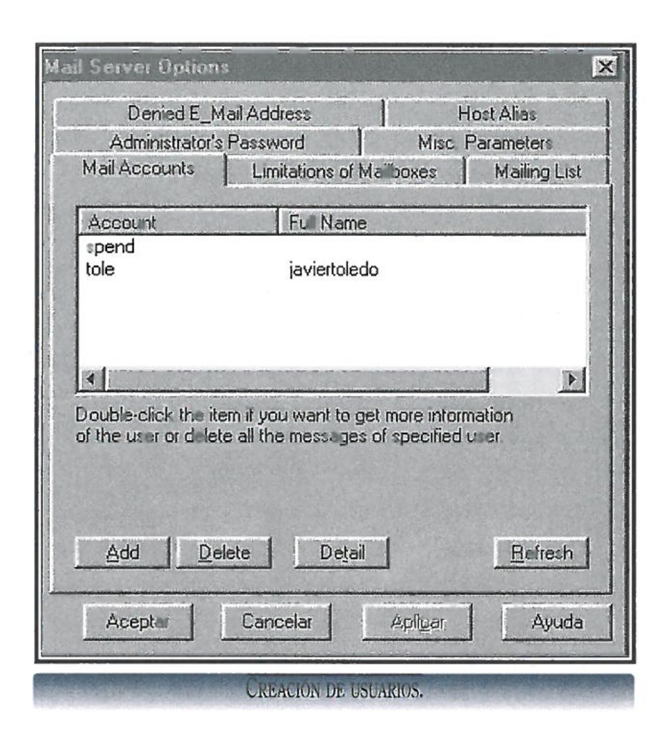
PROTOCOLOS

El SMTP es el protocolo que utilizan tanto el programa cliente, que utilizamos para mandar el mensaje, como los Mail-Router, que son los servidores locales de correo electrónico que se van pasando el mensaje hasta que llega al servidor en el que el destinatario del correo tiene su buzón de mail.

Para mandar el mensaje y transmitirlo mediante SMTP interviene otro protocolo llamado POP. POP3. Es la versión más utilizada y extendida de este protocolo en Internet. Gracias a este protocolo el programa de correo que estemos utilizando se chequea contra un Mail-Router identificándose mediante un usuario y una contraseña.

Si la verificación es positiva (el cliente tiene un buzón en el Mail-Router) el Mail-Router le envía los correos de la cuenta al programa cliente.

Un ejemplo de cuál es la manera más adecuada de utilizar el correo electrónico para transferencia de imágenes y archivos binarios es el protocolo MIME (Multi-purpose Internet Mail Extensions). El protocolo MIME, no sólo soporta una serie de formatos estándar, como son: imágenes GIF, BMP, JPG, ficheros postcript, ficheros de audio en formatos estándar, películas en formato MPEG, enlaces con conexiones vía FTP, páginas web, etc;, sino todo tipo de ficheros binarios.



CREACIÓN DE USUARIOS.

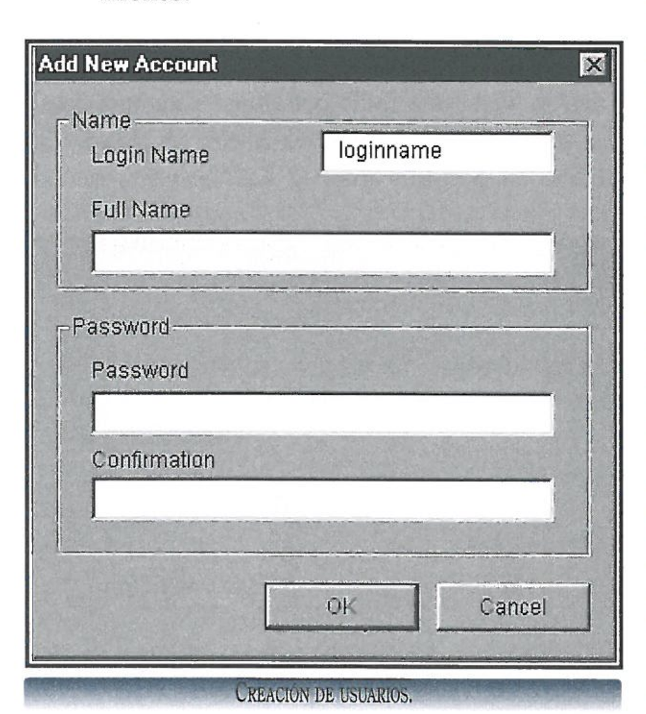
El programa de correo electrónico que va a enviar el mensaje convierte el archivo a una secuencia de caracteres ASCII. Gracias a la marca MIME se reconoce que el correo contiene un archivo binario codificado y al llegar a su destino se decodifica automáticamente.

ZETAMAIL

ZetaMail es un programa que combina SMTP y POP3 para ejercer como servidor de Mail bajo Windows 95 y/o NT. Funciona con cualquier cliente de correo como el de Netscape o el Internet mail de microsoft.

Características:

- Instalación Fácil.
- Provee SMTP y POP3 (RFC821 y RFC1725).
- Crea automáticamente un buzón para cada usuario de Windows NT.
- Tiene barreras que previenen a su sistema del ataque de un "mailbomb".
- Rechaza mensajes de algunas direcciones que le configuremos. Para evitar el uso abusivo por parte de algunas personas (Hay personas que se dedican a bombardear Internet con mensajes publicitarios).
- Permite el reenvio a la dirección del remitente.
- Permite crear listas de correo. Lo que nos permite que al enviar un mensaje de correo a una de estas listas el mensaje le llegue a cada una de las personas que pertenecen a la lista en cuestión.
- Alias de direcciones. Dos direcciones distintas van a parar a un mismo buzón.
- Respuesta automática. La respuesta automática nos permite configurar los buzones de tal manera que cualquier mensaje que llegue sea respondido automáticamente.



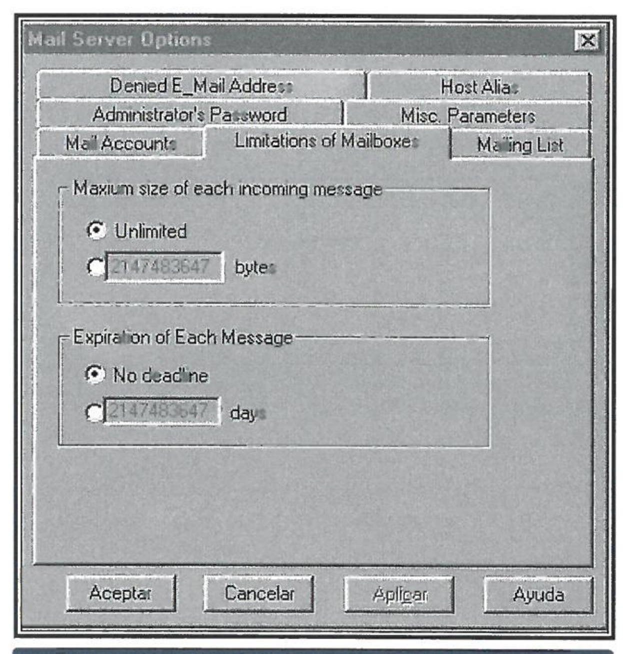
CREACIÓN DE USUARIOS.

Requerimientos.

- ❑ Microsoft Windows 95.
- ❑ Intel 80486 o superior.
- ❑ 16 + MB Memoria.

Crear nuevas cuentas

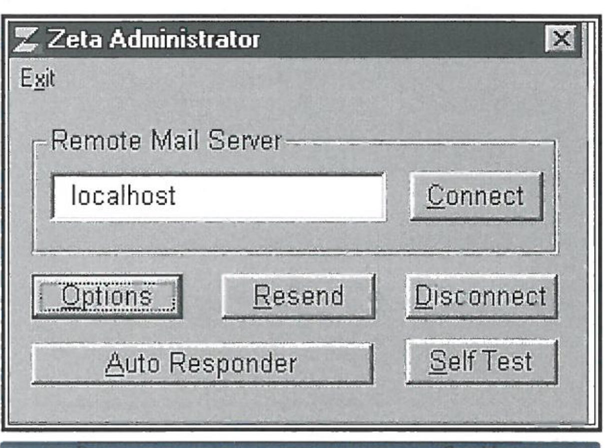
Para crear nuevas cuentas de correo tenemos que situarnos en la solapa de Mail Accounts y pulsar Add. Se abrirá una pantalla que nos pregunta el nuevo login y el password. Una vez creadas la cuentas, veremos los mensajes que hay en el servidor, que se almacenarán en el directorio del disco donde se ha instalado el mail server. La extensión para un mensaje es MSG. Desde la misma solapa, podemos borrar las cuentas, cambiarles la contraseña, borrar sus mensajes, etc.



PANTALLA DE CONFIGURACIÓN DEL TIEMPO DE VIDA DE LOS MENSAJES.

Crear nuevas listas de correo

Una de las ventajas más importantes de este servidor son las listas de correo. Podemos crearlas y manejarlas desde la solapa de Mailing List. Con la versión shareware sólo podemos crear una lista. Lo primero que hemos de hacer es activar la opción de permitir mailing list. Una vez hecho esto, creamos la lista introduciendo el título y la clave. Para que un cliente se añada a lista hay que enviar un correo con el asunto como "subscribe" a la dirección password@host (donde host es el nombre del ordenador y la password la introducida en la creación de la lista).



PANTALLA DESDE LA QUE SE LANZA EL TEST PARA ZETAMAIL.

Test

Cuando queremos estar seguros de que el servidor está funcionando, deberemos lanzar un Test que dirá si hay o no algún problema. Lo podemos hacer pulsando "Self Test" desde el administrador.

Configuración del tiempo de vida de los mensajes

Para configurar cuánto tiempo han de permanecer los mensajes en el servidor, lo podemos hacer desde la solapa de "Limitations of mailboxes". Desde aquí especificamos el tamaño máximo de cada mensaje y cuál es su tiempo máximo de vida.

Claramente, IBM es la compañía que ofrece más soluciones de software para Windows NT.

Conozca los valores de la familia de productos de software IBM para Windows NT.

Este gran puzzle de software para Windows NT funciona a la perfección. Aquí tiene todo lo necesario para desarrollar y desplegar aplicaciones, integrarlas con sus sistemas centrales, ampliarlas a su web site y gestionarlo todo. Y todas las piezas del puzzle encajan. Todas están preparadas para Internet desde que las extrae de la caja. Y por supuesto, mantienen el nivel de calidad que usted espera de IBM: Disponibilidad. Integridad. Escalabilidad. Soporte. Si desea ver más cerca cada una de las piezas y tener información detallada de ellas, visítenos en: www.software.ibm.com/nt, o www.ibm.es



Soluciones para nuestro pequeño mundo